МОНИТОР

журнал для программистов — практиков

1.93

ISSN 0869—3889

Вы боитесь сделать неверный шаг? ЮСИС всегда даст Вам правильный совет.

1. ЮСИС представляет собой программный комплекс, обеспечивающий поиск и ввод информации о нормативных актах, научно-практических комментариях к ним, судебной, арбитражной и нотариальной практике применения, типовых формах договоров, бланков и т.п. по всем отраслям права.

Программное обеспечение дает возможность пользователю вести собственную базу данных правовой и другой необходимой ему информации, включая ведение делопроизводства.

2. Система ЮСИС содержит ранее принятые нормативные акты и позволяет учитывать их изменения. При этом обеспечивается поддержка нормативного акта в контрольном состоянии.

3. Оперативное обновление информации, дружелюбный интерфейс системы, не требующий от пользователя специальных знаний компьютерной техники и юриспруденции.

ЮСИС как средство правовой информатизации используется Государственной налоговой службой России, Министерством промышленности и Министерством безопасности России, Внешторгбанком России и другими государственными и коммерческими организациями.

4. Программа функционирует в операционной системе MS-DOS, версия 3.3 и выше, в среде CLIPPER и адаптирована для работы на ПЭВМ, совместимых с IBM PC/XT, AT и PS/2 (Robotron 1910, Robotron 1834, ЕС 1841, Правец 16...).

Систему и ее сопровождение можно получить по почте или воспользовавшись модемной связью.

5. Если Вы заинтересованы в приобретении системы ЮСИС, мы вышлем Вам бесплатно демонстрационную версию программы, а также подробные правовые и технические характеристики системы вместе с проектом договора на абонементное обслуживание.

107120, Москва, пл. Прямикова, 5Тел.: 272-08-60, телекс: 411-700 SOCIN 003491, факс: (095)200-22-17 SOCIN 003491

Юсис—безупречная законность каждого Вашего шага.

INTRALEX

"Примерно с 1870 года начался всемирный бумажный потоп, продолжающийся и по сей день, причем по объему бумажной продукции рекламные агентства соперничают теперь даже с министерствами".

"Повседневный контроль над печатной индустрией крупный капитал осуществляет посредством коммерческой рекламы... 70-80% доходов печатные издания получают от публикации рекламных объявлений. Реклама занимает в среднем 60-70% объема газетных полос, журнальной площади".

Сирил Н. Паркинсон.
Законы Паркинсона. — М.: Прогресс, 1989.

Современные Соединенные Штаты Америки: *— Энциклопедический справочник. М.: Политиздат, 1988.*

В *первом номере "Монитора" за 1992 год с целью выявления интересов и взглядов наших читателей мы опубликовали анкету. Мы благодарим всех наших читателей, принявших участие в анкетировании (в течение года мы получили 207 заполненных анкет). Ваши ответы очень помогли нам в работе. Среди приславших анкету были разыграны призы; список победителей — на стр. 49.*

В анкету был включен и такой вопрос: "Какой объем должна, по Вашему мнению, занимать в современном компьютерном журнале реклама?" Результат удивителен: 90% наших читателей от Львова на западе до Петропавловска-Камчатского на востоке и от Севастополя на юге до города Мирный Архангельской области на севере, не сговариваясь, назвали одну и ту же цифру: 5% от объема журнала. 5 процентов! В 80-страничном журнале это — 4 страницы.

Рекламы в "Мониторе" будет, конечно же, гораздо больше, чем 4 страницы. Мы просим неприязненно относящихся к обилию рекламы читателей обратить внимание на три обстоятельства.

Первое. С увеличением количества компьютерных журналов и более четкой их дифференциации по темам и читательской аудитории реклама в каждом из журналов все более будет соответствовать интересам читателей конкретного журнала. В нашем журнале будет концентрироваться реклама компиляторов, отладчиков, инструментальных средств, библиотек и прочих весьма полезных для программиста вещей.

Второе. Рост объема рекламы будет, несомненно, сопровождаться и ростом объема, занимаемого техническими статьями. На сегодняшний день каждая дополнительная страница рекламы позволяет нам либо увеличить на 4 страницы объем журнала, либо поднять на 1,5 тысячи экземпляров тираж.

Третье. Все познается в сравнении. Чтобы понять, много или мало рекламы в "Мониторе", снимем с полки один из номеров популярнейшего американского журнала для программистов и сравним его с текущим номером "Монитора".

Журнал	Общий объем (стр.)	Технические статьи (стр.)	Из них листингов (стр.)	Реклама	Процент технических статей
Dr. Dobb's Journal № 180, 1991, сент.	160	73	24	87	46
Монитор, 1993, № 1	80	66	26	14	83

Как Вы видите, при вдвое меньшей толщине в "Мониторе" примерно такой же объем содержательной информации.

Кстати, о содержании. В 1993 году "Монитор" станет еще более "программистским" журналом, чем прежде. Вследствие увеличения объема номера мы сможем как выделить несколько страниц для начинающих программистов (хотя и не будем заниматься ликбезом), так и предоставить место профессионалам со статьями, требующими для понимания достаточно высокой квалификации. Мы хотим, чтобы, во-первых, журнал был интересен всем программистам, и чтобы, во-вторых, прочитав сегодня треть или половину журнала, Вы могли бы вернуться к нему через несколько месяцев (не лет, прогресс идет достаточно быстро!).

А чтобы сгладить огорчение наших читателей-пользователей по поводу исчезновения из "Монитора" предназначенных для них статей, мы сообщаем о том, что начинаем издание журнала о современных компьютерных технологиях для пользователей. Название журнала (предположительное!) — "Ориентир". Первый его номер выйдет в свет в апреле 1993 года. Все читатели "Монитора", которые вырежут купон на стр. 34 и пришлют его в редакцию до 15 мая 1993 года, получат этот номер нового журнала БЕСПЛАТНО.

С пожеланиями трудовых свершений

Редакция журнала "Монитор"

Журнал для программистов-практиков

Редколлегия:
Д.Э.Бройтман
(главный редактор)
Р.А.Болдырев
А.А.Винокуров
(отдел распространения)
В.В.Водолазский
Е.В.Гун
Г.Л.Карчикян
(компьютерная верстка)
С.В.Макельский
(главный художник)
А.М.Миляков
(ответственный секретарь)
Н.А.Никитина
(отдел рекламы)
А.В.Орехов
(издательский директор)
Л.И.Турок

Представительство в Европе (Германия)
Марина Гетманская
тел. 8-(0631)-7-49-05

В оформлении номера использованы рисунки
А.С.Замулы

Подписано в печать с оригинал-макета 01.02.93
Формат 60x84/8.
Тираж 30 000 экз.
Печать офсетная. Печ. л. 10,0
Изд. № 1971.
Цена договорная
Отпечатано в ИПК "Московская правда" 123845, Москва, Д-22, ул. 1905 года, д. 7 Заказ 0493
Издатель:
Научно-техническое общество
Софт-Москва

Для писем: 117419, г. Москва, а/я 765
Тел. (095) 238-99-33
Факс (095) 238-32-78

© "Монитор", 1993

Технологии программирования

А. Петренко. **Познакомимся с VDM** **4**

Венский метод разработки программ (VDM) — это набор средств для формального специфицирования программных комплексов и методология, которая на базе разработанных спецификаций позволяет конструировать надежные программные системы.

С. Кириллов. **Скорость выполнения и размер кода программы. Возможен ли компромисс?** **10**

А. Кельманзон. **Объектно-ориентированное программирование на BORLAND C++: контроль распределения памяти при отладке** **12**

В статье описан метод, позволяющий контролировать динамически распределяемую память в C++, не внося изменений в исходный текст программы.

Д. Арапов. **Пишем упаковщик** **16**

Обсуждаются методы упаковки текстов, в частности, алгоритм Лемпела-Зива.

Д. Солдатенков. **Трехмерная графика** **26**

Статья посвящена созданию трехмерных изображений на экране дисплея. Приведен пример программы на языке Turbo Pascal 6.0, отображающей поверхности вида Z = F (x, y) методом трассировки обратного хода луча.

Базы данных

Л. Ливеровский. **Модули расширения языка Clarion** **31**

Рассматриваются назначение и структура модулей расширения языка Clarion (LEM — language extension module). Приводятся примеры программирования LEM на языке ассемблера и C.

Игры

Adventure Games: эволюция жанра Часть 2 **38**

Во второй части статьи, посвященной анализу адвентюрных игр, рассматривается использование в играх графических возможностей персональных ЭВМ.

Сети

Ком-СТЕЛС — телекоммуникации в России **42**

Базы данных

И. Кравчинский. **Возможности СУБД db_VISTA III** **44**

В статье описывается СУБД db_VISTA 3.5. Приведен пример использования db_VISTA 3.5 для реализации простой базы данных на языке С.

А. Сунгатуллин. **Мини-СУБД в Вашей программе** **51**

Статья посвящена вопросам организации поддержки файлов прямого доступа из программ на языке С. Приведены исходные тексты библиотеки Minibase и пример реализации телефонного справочника.

Аппаратура

"НАТЕКС" — наступление продолжается! **59**

Статья рассказывает о Hayes совместимых модемах, предлагаемых фирмой "Натекс".

DOS под микроскопом

А. Аброськин. **Снова о TSR или как пользоваться прерываниями DOS и BIOS из Вашей резидентной программы** **60**

Описан способ организации взаимодействия резидентных программ с операционной системой. Язык программирования — Turbo Assembler.

Право

Законы надо уважать! **66**

Компьютер в разрезе

Б. Телеснин. **Адаптер VGA. Режим 256 цветов** **67**

Описаны регистры контроллера, их программирование. Приведен пример на языке Turbo Pascal.

Экспертные системы

Ю. Носков. **Практика построения экспертных систем средствами оболочки VPX** **76**

Оболочка VPX — простое и удобное средство для построения несложных экспертных систем (ЭС); на базе оболочки могут быть успешно построены прототипы более сложных ЭС.

Познакомимся с VDM

А. Петренко

VDM (Vienna Development Method - Венский метод разработки программ, [1]) - это совокупность формальных методов и некоторых правил (рекомендаций), позволяющих на достаточно строгой основе конструировать "правильные" программные системы. VDM можно рассматривать как развитие на новой основе методов проектирования сверху вниз, пошаговой детализации, абстрактных типов данных. Причина успехов VDM в удачном соединении методики пошагового уточнения с формальным и строгим описанием логических моделей системы, постепенно уточняющихся от шага к шагу, и формального доказательства корректности этих уточнений.

VDM возник на базе работ Венской лаборатории фирмы IBM, занимавшейся методами формального описания языков программирования и создания компиляторов на основе таких описаний. В свое время среди специалистов по теории формальных языков получил некоторую известность VDL (Vienna Definition Language) - венский язык. В других областях программирования он не использовался. Судьба VDM сложилась счастливей. Он получил значительно более широкое распространение, и, кроме того, оказалось, что, в отличие от VDL, может использоваться не только при разработке языков и компиляторов, но и при создании многих самых разнообразных программных и программно-аппаратных систем.

В кратком изложении суть VDM можно изложить следующим образом. Проектирование программной системы разбивается на шаги. На первом шаге создаются исходные спецификации системы, которые затем уточняются. Спецификация описывает (быть может, в весьма общих чертах, но строго и полно для выбранного уровня детализации) структуру системы, взаимозависимости между отдельными компонентами и операции над данными структурами. Взаимозависимости описываются при помощи предикатов, задающих некоторые инвариантные свойства. Эти свойства определяют, что составляет существо корректного состояния данной системы, например, согласованность различных каталогов, счетчиков, полноту списков и тому подобное. Считается, что инварианты выполняются перед обращением к любой из операций и после ее выполнения. Определением операции служат пред- и постусловия. Первые ограничивают условия, при которых возможно обращение к операции, вторые, по существу, определяют результат: что дает операция или какие изменения в модели системы она порождает.

Для описания структур данных, инвариантов и операций используется язык META-IV. После того как первый шаг завершен, выполняется шаг уточнений спецификаций. В конечном итоге задача разработчиков — от спецификаций на языке META-IV перейти к текстам программ на некотором языке программирования. Поэтому шаг уточнения состоит в замене некоторых сложных структур в спецификациях на структуры данных, достаточно просто реализуемые в выбранном языке программирования.

Строго говоря, на каждом переходе требуется формально доказывать корректность уточнения. На практике это не всегда удается (пожалуй, даже почти никогда в полном объеме не удается). Однако теория указывает на те моменты, которые требуют доказательства корректности, а они-то, как показывает практика, чаще других являются источниками ошибок. Знание таких критичных мест облегчает анализ и помогает своевременному обнаружению ошибок.

Роль языка Meta-IV при проектировании программ можно сравнить с ролью языка чертежей в машиностроении. И тот и другой позволяют выполнить как эскизный набросок, так и подробный проект. Но заметим, что помимо "языка чертежей" VDM дает и общую схему их анализа, критерии оценки и методы последовательной детализации.

Из наиболее известных примеров успешного применения VDM известен опыт создания компилятора языка Ада под руководством Д.Бьернера (D.Bjorner). Он был разработан примерно за три года и практически не содержал ошибок. Для таких сложных систем, как компилятор с Ады, это, конечно, рекордный результат. Вместе с тем, в публикациях появляются сведения и о других успешных применениях VDM. Так, несколько фирм Великобритании взяли VDM за основное средство проектирования программных систем и описания спецификаций.

VDM является весьма эффективным средством повышения качества и сокращения сроков разра-

боток. Это достигается, во-первых, за счет значительного внимания, уделяемого этапу проектирования, во-вторых, VDM позволяет естественным образом организовать разделение труда в коллективе программистов. Более квалифицированные разработчики составляют спецификации и контролируют процесс программирования,опережающеепоявлениеспецификаций позволяет привлекать к программированию и составлению тестов достаточно большое количество разработчиков.

VDM не имеет специальных средств для описания параллельных процессов или реального времени. Это можно рассматривать как одно из ограничений метода, однако не следует думать, что VDM бесполезен в этих областях. Сфера "промышленного" использования, как это ни парадоксально, в основном захватывает фирмы, которые занимаются созданием больших систем реального времени, систем навигации и других разработок. Несмотря на отсутствие специальных средств описания параллельности, что является существенной особенностью этих систем, VDM полезен в этих разработках как средство повышения надежности.

Необходимо отметить ряд проблем, возникающих при использовании VDM. Самая главная трудность - это использование нетрадиционного для опытных программистов языка и стиля проектирования. VDM легко овладевают студенты младших курсов вузов и люди, имеющие склонность к математически строгим, формальным построениям. Но если относиться к VDM без излишней предвзятости, то пользу из его применения сможет извлечь для себя каждый.

Наша цель только познакомиться с VDM: полное описание языка спецификаций VDM Meta-IV и собственно метода требует обстоятельной монографии. Поэтому мы выбираем путь разбора иллюстративного примера. Надеемся, пример будет понятен большинству читателей. Однако не забывайте, что в жертву "популизму" мы приносим некоторые важные возможности VDM, в частности, средства описания языков и методики разработки компиляторов на их основе.

Итак, пусть нам предстоит создать систему для работы с генеалогической информацией - родословной (в каждой семье надо иметь такой архив). Венский метод предписывает сначала разработать эскизный проект, затем, если необходимо, уточнить проектные спецификации и лишь затем перейти к реализации.

На каждом из шагов разработки необходимо
- определить набор понятий, в терминах которых будут специфицироваться структуры данных и операции,
- определить структуры данных,
- определить инвариантные свойства структур данных,
- описать операции и доказать, что выбранные спецификации гарантируют сохранение инвариантов.

Для описания генеалогии понадобятся следующие базисные понятия:

```
Name  —  имя — некоторая строка,
         позволяющая однозначно
         идентифицировать личность,
Sex   -  пол личности
```

Этими базисными понятиями мы и ограничимся. Годы жизни и смерти, место рождения и другие сведения в нашей системе мы не храним. (Заметьте, что если этих сведений нет в эскизе, то они не имеют права появиться и в других спецификациях системы). Тем самым личность у нас будет иметь ровно две характеристики

```
Person  ::  имя  :  Name
            пол  :  Sex
Name    =   Token
Sex     =   Quot
```

Комментарий к обозначениям

Мы описали типы данных Person, Name и Sex. При помощи знака "::" описываются деревья, структуры, подобные записям Паскаля, тип Token описывает любые строки символов, Quot - что-то близкое к перечислимому типу данных, при этом, однако, не требуется уточнять, какие именно имена входят в этот тип. Кроме Token и Quot в VDM в качестве базисных используются Bool (булевский тип), Nat0 и Nat1 (натуральные числа с нулем и без нуля) и Int (целые).

Теперь зададимся вопросом, как представлять родственные связи в нашем архиве. Пока мы располагаем только одним средством составления композиции - деревом. В VDM есть еще три возможности: множества, списки и отображения (последний способ рассмотрим чуть позже). Если речь идет об определении некоторого типа данных-множества, составленного из элементов типа X, то этот тип, назовем его S, описывается так:

$$S = X\text{-set}$$

Тип список из элементов типа X описывается как

$$L = X^* \text{ или } L = X^+$$

("*" означает, что списки могут быть и нулевой длины, "+" означает, что списки не пустые, а содержат хотя бы один элемент).

Между множествами и списками имеется два принципиальных отличия:
- во множестве все элементы уникальны, в списке один и тот же элемент может встречаться несколько раз;
- элементы множества не упорядочены, в списке элементы нумеруются.

Заметим, элементами списков и множеств, так же как значениями компонентов дерева, могут быть объекты любого вида (как базисного, так и небазисного типа).

Однако читатель должен обратить внимание, что "изюминка" VDM - не структуры данных и не нотация, которая используется для спецификаций, а собственно метод, "метода", следуя которой, разработчик добьется поставленных целей. Первая заповедь "методы" - разбивка процесса проектирования на шаги, о которых шла речь раньше. Вторая заповедь предписывает крайне тщательно разрабатывать все, особенно первые, эскизные спецификации, ведь известно, что ошибки первых фаз жизненного цикла самые дорогие.

Тщательность требуется и в большом, и в малом. Проектные спецификации можно рассматривать как модель будущей системы. От модели требуется, чтобы в ней были представлены, и достаточно естественно, все основные составляющие системы, ее операции, характерные атрибуты и свойства. Кроме того, в модели не должно быть ничего лишнего. Для эскизной модели это означает, в частности, что в ней не должно быть ничего, что связано с особенностями реализации. Заметим, что для программиста отрешиться на время от проблем реализации бывает порой очень тяжело.

Вернемся к нашему примеру. Как составить композицию из объектов типа Person для представления генеалогического дерева (которое, если приглядеться более пристально и рассмотреть все связи, и не совсем дерево) (рис. 1).

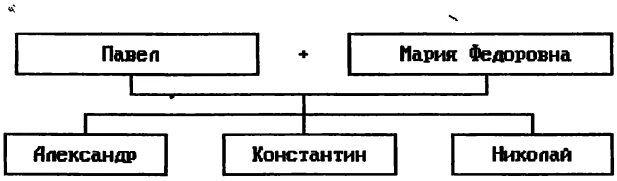


Рис. 1

По-видимому, первой нас посетит идея (в общем-то неплохая) построить граф при помощи попарных связей:

Муж —> Жена или Супруг —> Супруг
Родитель —> Ребенок или Ребенок —> Родитель
и тому подобное

В расширенном варианте могут быть добавлены связи

Брат(сестра) —> Братья(сестры)
Ребенок —> Отец
Ребенок —> Мать и другие

Для начала расширим тип Person двумя связями: "супруг" и "дети", - и остановимся на таком варианте модели:

```
DB     =  Person
Person ::  имя    :  Name
           пол    :  Sex
           супруг :  [Person]
           дети   :  Person*
```

То есть все содержимое базы данных доступно через корень дерева родственных связей. Все узлы этой структуры однотипны. В каждом узле, помимо имени и пола, хранятся ссылки на узел, представляющий супруга (мужа или жену), и список детей. Квадратные скобки обозначают, что связь "супруг" может быть пустой. Почему при описании ветви дерева "дети" мы использовали список, а не множество? Потому что, вероятно, в системе важно хранить информацию о том, какой ребенок старше, а какой младше. Мы не забыли указать, что список может быть и пустым - из таких деталей и складывается адекватность модели решаемой задаче.

В системе связей между элементами не предусмотрена связь Дети —> Родители. Это обычный прием, такая связь являлась бы избыточной, дублирующей уже имеющуюся информацию, от нее можно отказаться. Но допустим, что мы все же ввели атрибут

родители : Person$^+$,

полагая, что по такой связи можно получить доступ к списку из двух узлов типа Person, представляющих мать и отца данного человека. Если мы сохраним этот атрибут в данном виде, то мы внесем в проект, по крайней мере, одну ошибку и, возможно, несколько неясных моментов, что в конечном итоге также можно рассматривать как проектные ошибки.

Во-первых, к сожалению, не всегда есть сведения о родителях ребенка, поэтому следует писать хотя бы так:

родители : [Person$^+$] родители : Person*

Во-вторых, избрав для представления родительской пары форму списка, мы навязали модели предметной области некоторое свойство, которое ей не присуще. Вполне резонно задать, например, такие вопросы:
- кто стоит на первом месте, мать или отец?
- может ли один и тот же объект быть элементом такого списка?

Появление такого рода вопросов говорит о том, что здесь в проекте неясное место, кроме того (что может быть в случае другой задачи очень важно), требование хранить информацию о каких-либо объектах в упорядоченном виде может привести к неоправданно большим дополнительным расходам.

Посмотрим, не подойдет ли для данной цели множество

родители : Person-set

Множество может быть пустым - это будет соответствовать случаю, когда сведения о родителях отсутствуют. Во множестве нет никакой упорядоченности, и один и тот же объект не может дважды входить в одно множество, следовательно, такой атрибут мог бы расширить нашу модель вполне адекватно решаемой задаче.

Теперь познакомимся с еще одной возможностью конструирования структур данных - отображениями. Отображения (map) можно рассматривать как таблицы с двумя столбцами. Левый столбец состоит из ключей, правый - из некоторых значений, связанных со своими ключами. Как ключи, так и значения могут быть объектами любого типа, при описании структуры таблицы необходимо объявить эти типы. Так, например, тип хеш-таблицы (HT), которая позволяет по содержимому текстовой строки (Token) получить некоторый код - натуральное число (Nat0), можно было описать так

HT = Token –m–> Nat0

Посмотрим, не пригодится ли такая структура данных в нашей генеалогической базе данных. Но сначала попробуем найти изъяны в уже имеющемся варианте описания структуры.

Представления родословного дерева в виде набора узлов, соединенных между собой попарными связями, было подсказано нам графическим образом, к которому мы привыкли, - так рисуют родословные на бумаге. Однако,

Glossary

ГЛОССАРИЙ ОСНОВНЫХ ОБОЗНАЧЕНИЙ ЯЗЫКА META–IV

Для каждого вида данных описываются средства для определения доменов, объектов и операций над ними.

Boolean

Bool	= { false, true }			
операции		аргументы		результаты
¬	отрицание	Bool		Bool
&	и (конъюнкция)	Bool	Bool	Bool
V	или (дизъюнкция)	Bool	Bool	Bool
⇒	следует (импликация)	Bool	Bool	Bool
=	равно	Bool	Bool	Bool
∀	для всех		* * * * * * *	
∃	существует		* * * * * * *	

Арифметика

Int	= { . . . -1,0,1, . . .}
N0 или Nat0	= {0,1,2, . . .}
N1 или Nat1	= {1,2,3, . . .}

операции

+, -, *, /, **, =, <, > и т.д.

Строки

Quot — множество перечислимых элементарных (атомарных) текстовых объектов (строк)

операции

=	равно
≠	не равно

Символы (Tokens)

Token — множество элементарных объектов, представление которых не рассматривается

операции

=	равно
≠	не равно

Множества

–set	определяет все конечные подмножества данного множества
{a,b,c}	явное определение множества
{a ∣ P(a)}	неявное определение
{a ∈ S ∣ P(a)}	неявное определение

операции		аргументы		результаты
∈	принадлежит	X	X-set	Bool
¬∈	не принадлежит	X	X-set	Bool
∪	объединение	X-set	X-set	X-set
∩	пересечение	X-set	X-set	X-set
–(\)	разность	X-set	X-set	X-set
⊂	строгое включение	X-set	X-set	Bool
=	равно	X-set	X-set	Bool
≠	не равно	X-set	X-set	Bool
card	мощность	X-set		Nat0
union	дистрибутивное объединение	(X-set)-set		X-set

Списки (Tuples,Lists)

*	все конечные списки из элементов данного множества, включая и пустой список
+	то же, но без пустого списка
<a,b,c>	явное задание списка
<f(i) ∣ P(i)>	неявное задание

операции		аргументы		результаты
hd	голова списка (1-ый элемент)	X^+		X
tl	хвост списка (tail)	X^+		X^*
[ ] или ( )	указание i-го элемента	X^+		X
len	длина	X^+		Nat0
inds	множество индексов (номеров элементов данного списка)	X^+		Nat1-set
elems	множество, составленное из элементов данного списка	X^+		X-set
^	конкатенация	X^+	X^+	X^+
=	равно	X^+	X^+	Bool
≠	не равно	X^+	X^+	Bool
conc	дистрибутивная конкатенация		$(X^+)^+$	X^+

Деревья (Trees)

A :: . . .	определяет домен A вида Tree
*	определяет декартово произведение — неименованное (или именованное) дерево

операции

mk-A(. . .)	конструктор, при помощи которого можно явно описывать объекты типа A

Glossary

Глоссарий (продолжение)

s-fl(O)	функция-селектор, которая позволяет получить ветвь (поле) s-fl объекта O
=	равно
≠	не равно

Отображения (Maps)

–m–>	определяет все конечные отображения между данными множествами
<–m–>	определяет взаимнооднозначное отображение
[a —> x,b —> y]	явное задание отображения
[d —> f(d) ∣ P(d)]	неявное задание отображения

операции		аргументы		результаты
Пусть M = A –m–> B				
( )	применение (получение значения по данному ключу, например, M(d))	A		B
∪	объединение (слияние)			
+	слияние с наложением	M	M	M
\	удаление	M	A-set	M
∣	ограничение	M	A-set	M
dom	домен (множество ключей)	M		A-set
rng	множество значений	M		B-set
=	равно	M	M	Bool
≠	не равно	M	M	Bool
merge	дистрибутивное	M-set		M

Аппликативные конструкции для определения функций

let id=e in e2	блок-выражение; определяет значение свободных вхождений id
let mk-T(a,b, . . .)=e1 in e2	
f(x)	применение (вызов) функции

Императивные конструкции для определения функций

dcl v := e type D	объявление переменной типа D
c	оператор, указывающий на содержимое некоторой переменной
v := e	присваивание
;	композиция (сочленение) операторов
def Id : e; s	императивный let
while e do s	цикл while
for i=m to n do S(t)	итеративный цикл
for all a ∈ Set do S(a)	итеративный цикл с произвольным порядком использования элементов из Set
return(t)	выдача "императивного значения"
if t then c else a	если . . . то . . . иначе . . .
b1 —> c1, . . . bn — cn,	if-оператор с несколькими ветвями
cases b0: b1 —> c1,. . .,bn —> cn,	case-оператор, где b0 - выражение, a bi (0 < i < n+1) один из типов, который может иметь выражение b0

если вдуматься, то в предложенной модели есть ряд неудобств. Во-первых, для доступа к произвольному узлу всегда придется пройти путь от корня дерева до данного узла, то есть модель навязывает структуре некоторую упорядоченность хранения (и доступа), которая для нашей задачи будет удобной далеко не всегда. Во-вторых, во всех узлах предусмотрена связь с супругами и детьми, а для многих узлов такие связи не нужны, тем самым здесь есть резерв для экономии памяти.

Основная идея нового варианта структуры - хранить информацию о родственных связях в явном виде и описывать их при помощи отображений. Например, так:

Муж_жена = Name –m–> Name

Для объектов типа отображение определены операции, которые позволяют получить множество ключей (dom), множество значений (rng), слить два отображения в одно (∪ или +), исключить из отображения часть строк (\). Для задания значений типа отображение используется следующая нотация

[x1 —> y1, x2 —> y, ...]

Здесь x1 и x2 представляют значения ключей, а y1 и y2 - значения, связанные с соответствующими ключами.

Пусть имеется отображение для хранения номеров телефонов. Его тип можно определить так:

```
Quot -m-> Nat1
```

Продемонстрируем действие упомянутых операций .

```
dom [Петров—>1250062,Иванов—>5328331] = {Петров,Иванов}
      — домен пустого отображения — пустое множество
dom [ ] = { }
rng [Петров—>1250062,Иванов—>5328331] = {1250062,5328331}
[Сидоров—>1111111] ∪ [Петров—>42424] =
      [Сидоров—>1111111, Петров—>42424]
[Сидоров—>1111111] + [Петров—>42424, Сидоров—>222 ] =
      [Сидоров—>222, Петров—>42424]
```

Операция + в случае сложения отображений выполняет слияние с наложением, старое значение ключа изменяется на новое. Для операции ∪ пересечение множества ключей в левом и правом отображении запрещено.

```
[Сидоров —> 111, Петров —> 444, Иванов —> 333] \
    {Петров, Сидоров} = [Иванов —> 333]
```

Вернемся к задаче о генеалогической базе данных. Для того чтобы одинаково понимать, что же требуется от нашей БД, нужно четко сформулировать задачу; постараемся сделать это достаточно кратко.

БД должна хранить сведения об отдельных лицах и их родственных связях. Личность однозначно идентифицируется именем, известен пол каждой личности. Имя и пол - неизменяющиеся характеристики личности.

В произвольном (но разумном) порядке в БД могут вводиться сведения о родственных связях. БД хранит сведения о браках и о том, кто чьим ребенком (родителем) является. БД для каждого лица может хранить сведения не более, чем об одном браке.

БД должна обеспечить ввод информации о рождении, заключении и расторжении браков и вывод в форме распечатки родословного дерева для указанного лица.

Попробуем применить для спецификации нашей БД отображения. Будем в явном виде хранить сведения о родственных связях и откажемся от графа с попарными связями. Тогда структура БД будет выглядеть так:

```
NewDB   ::  пол              :  Name  -m->Sex
            браки            :  Name  -m->Name
            ребенок-родители:  Name  -m->Name-set
```

Похоже, что в данной структуре мы сумели избавиться от тех недостатков, которые были в прежних вариантах.

Пора переходить к описанию инвариантных свойств (инварианта) нашей структуры. Сначала сформулируем их неформально.

1) Каждый, у кого зарегистрирован брак или имеются сведения о родителях или детях, обязательно зарегистрирован в отображении "пол".

2) Если есть сведения, что некоторое лицо состоит в браке с другим, то такие же сведения хранятся и для его супруга (супруги).

3) В браке состоят только лица разного пола.

4) Родителей не более, чем двое, и они разного пола.

5) Человек не может фигурировать в списке своих предков.

Средства VDM позволяют описывать такие инварианты формальным образом. Опыт показывает, что работа над формулировкой инварианта (даже на неформальном языке) - это очень полезное занятие. В процессе придумывания инварианта, как правило, выявляются детали описания структуры, которые привносят неоправданную громоздкость, а то и ошибки. Когда мы переписываем инварианты формально, мы как бы анализируем структуры с еще одной позиции и видим ее в новом ракурсе. Это нередко позволяет вскрыть и новые недостатки.

По форме инвариант - это предикат, функция, результатом которой служит булево значение. Аргумент инварианта - состояние системы. Инвариант обычно представляет собой конъюнкцию - перечисление свойств, которые должны выполняться для любого состояния системы. Система находится в корректном состоянии тогда и только тогда, когда инвариант истинен. Номера строк мы будем использовать в комментариях, так проще ссылаться на то или иное место в формальном тексте (пример 1).

Пример 1

```
1   inv-NewDB (d) =
2   (∀ p ∈ (dom d.браки ∪ dom d.ребенок_родители ∪
3         union rng d.ребенок_родители))
4   (p ∈ dom d.пол)                                    &
5   (∀ p ∈ dom d.браки)
6   ((d.браки(p) ∈ dom d.браки)&(d.браки(d.браки(p))=p))&
7   (∀ p ∈ dom d.браки)
8       (d.пол(p) ≠ d.пол(d.браки(p)))                 &
9   (∀ p ∈ dom d.ребенок_родители)
10      ((card d.ребенок_родители(p) ≤ 2) &
11          (∀ p1,p2 ∈ d.ребенок_родители)
12              ( p1 ≠ p2 => d.пол(p1) ≠ d.пол(p2)))   &
13  (∀ c ∈ dom d.ребенок_родители)
14      (c ¬∈ предки(d.ребенок_родители, {c}))
15  предки : (Name -m-> Name-set) Name-set—>Name-set
16  предки (mm, pp) =
17      {prnt p ∈ dom mm ∩ pp,
18          q ∈ (mm(p) ∪ предки(mm, mm(p)));
19          prnt = q }                                 ■
```

Комментарии

2-4. Свойство 1. Функция union называется дистрибутивным объединением, она, так сказать, раскрывает скобки в множествах, состоящих из подмножеств, например:

```
union  {{Петр_1,Евдокия}, {Павел, Мария_Федоровна}} =
       {Петр_1, Евдокия, Павел, Мария_Федоровна}
```

Поскольку с текстом на Meta-IV мы встречаемся по существу впервые, приведем словесный эквивалент.

Для любого лица p, которое входит в домены (является ключом) отображений "браки" или "ребенок_родители" или имеется в дистрибутивном объединении множества значений отображения "ребенок-родители" (то есть во множестве всех родителей), справедливо, что p принадлежит домену отображения "пол".

5-8. Свойства 2 и 3.

9-12. Свойство 4. Функция card вычисляет количество элементов во множестве, являющемся ее аргументом.

13-14. Свойство 5. Формулировка свойства проста за счет использования вспомогательной функции, которая определяется ниже.

15-19. Определение вспомогательной функции "предки", ее результат - множество, в котором перечислены имена всех прямых и непрямых предков указанного лица.

15. Описание типа функции, то есть типа ее аргументов (состояние БД и имя) и типа результата (множество имен).

16. Заголовок функции, mm и pp - формальные параметры, mm - отображение, pp - подмножество с именами родителей.

17-19. Самое непростое место в определении данной функции, вместо него можно было бы написать обычный алгоритм обхода дерева предков, но так, пожалуй, короче.

Теперь определим операции. Пусть их будет четыре, три для ввода информации и одна для выдачи родословных. Определим их полуформально, а для двух из них приведем и полное формальное определение. Общая схема определения операций следующая: сначала описывается тип аргументов и результатов, затем пред- и постусловия.

Предусловия (вместе с инвариантом!) полностью описывают условия применимости данной операции. При реализации необходимо решать, кто будет обеспечивать проверку выполнения предусловий, это может быть вызывающая программа, а может быть и сама программа операции. Постусловие описывает состояние системы после выполнения операции, при этом оно опирается на значения аргументов: состояние системы (d) перед выполнением и после выполнения операции (d').

Регистрация рождения

Рождение : NewDB Name Sex (Name | Nil) (Name | Nil) —> NewDB
— в четвертом и пятом параметрах может быть имя или Nil — признак того, что имя неизвестно.

pre-Рождение (d, n, s, m, f) =
— рождение n до этого не регистрировалось,
— если указано имя матери и мать зарегистрирована в отображении "пол", то ее пол женский,
— если указано имя отца и отец зарегистрирован в отображении "пол", то его пол мужской,
— n не совпадает с m и f и не фигурирует в предках своих родителей,
— если пол n уже зарегистрирован, то он совпадает с s,
— m и f различны.

post-Рождение (d, n, m, f, d') =
— в d' хранятся все сведения из d и добавлены сведения о том, что
— пол n равен s,
— m и f - родители n, и пол m женский, а пол f мужской.

Регистрация брака

Брак : NewDB Name Name —> NewDB

pre-Брак (d, w, h) =
— для w либо нет сведений о поле, либо это женщина,
— для h либо нет сведений о поле, либо это мужчина,
— w и h ни с кем не в браке
((w ¬∈dom d.пол) V
(d.пол(w) = "ЖЕН")) &
((h ¬∈dom d.пол) V
(d.пол(h) = "МУЖ")) &
((w ¬∈dom d.браки) & (h ¬∈ dom d.бра"и)

ELITSoft

Пользователям вычислительной техники Акционерное общество "ЭЛИТ СОФТ" предлагает

Агрегаты Бесперебойного Питания (Uninterruptable Power Source)

Основные характеристики

Мощность	900 ВА
Время автономной работы	40 мин.
Время переключения	5-7 мсек.
Рабочий диапазон	180-240 В
Гарантия	1 год
Послегарантийное обслуживание	

Поставка и оплата

Поставка со складов в гг. Москва, Санкт-Петербург, Чита, Рязань
Любая, удобная Вам, форма оплаты

В конструкции предусмотрены

Фильтрация сетевых помех
Звуковая и световая сигнализация
Индикация перегрузки
Индикация уровня заряда батарей
Герметичные, необслуживаемые аккумуляторные батареи

Адрес: 105215, Москва, 11-я Парковая, 44, корп. 1

Телефон
Секретариат: (095) 965 3529, 532 9614
Отдел сбыта: (095) 965 3528, 965 3530
Факс (095) 965 3528, 965 3530
Телетайп 205211 КТУР

Возможна доукомплектация компьютерами

```
post-Брак (d, w, h, d') =
    — в d имеются сведения о поле w и h,
    — имеются сведения о браке,
    — информация о детях и родителях не изменяется.
    d'.пол = d.пол + [w —> "ЖЕН"] + [h —> "МУЖ"]  &
    d'.браки = d.браки + [w —> h] + [h —> w]      &
    d'.ребенок_родители = d.родитель_ребенок

                Регистрация развода

Развод : NewDB Name Name —> NewDB

pre-Развод (d, w, h) =
    — w и h - супруги, этого достаточно.
    (w ∈dom d.браки) & (d.браки(w) = h)

post-Развод (d, w, h, d') =
    d'.пол = d.пол
    — сведений о браке w и h в базе данных нет.
    d'.браки = d.браки \ {w, h}
    d'.ребенок_родители = d.родитель_ребенок

                Напечатать родословную

Печать_родословной : NewDB Name —>

pre-Печать_родословной (d, n) =
    — в базе данных должна быть хоть какая-нибудь
          информация о n.
    (n ∈ dom d.пол)
```

На этом мы завершаем иллюстративный пример. Чего мы не сделали на первом шаге проектирования? Мы не доказали, что при данных определениях операций инвариант системы не нарушается. Проведя такое доказательство, можно было бы переходить к уточнению спецификаций. Первое, что потребовалось бы сделать, это выбрать, как на нашем языке программирования будут реализовываться отображения, это могут быть списки, индексные файлы и тому подобное. Выбрав новую систему понятий и структуру, мы должны были бы сформулировать новый инвариант и дать новые определения операций, доказать их корректность. После этого надо было бы доказать, что уточненные спецификации согласуются с эскизными. После одного-двух уточнений мы должны получить программу на языке программирования. Вот и вся работа. Как ни странно, иногда все это можно проделать и, более того, некоторые даже проделывают (например, программисты из команды Д.Бьернера). Пример таких уточнений и доказательств можно увидеть в [2].

Хотелось бы надеяться, что читатель, дочитавший статью до конца, готов разделить уверенность автора в том, что VDM в действительности является мощным средством проектирования, пригодным практически для всех областей разработки программ. Разработка программ по технологии VDM, возможно, является более трудоемкой по сравнению с традиционной, но уровень надежности при этом может быть существенно повышен - это средство для разработки дорогих и хороших программ. ■

ЛИТЕРАТУРА

1. **Bjorner D., Jones C.B., editors. Formal Specification and Software Development.- Prentice-Hall Int., 1982.**
2. **Петренко А.К. Венский метод разработки программ (Неформальное введение)// Программирование. 1991.- N 6.-С . 3-23.**

Скорость выполнения и размер кода программы. Возможен ли компромисс?

Очень часто при разработке программного обеспечения приходится искать компромисс между двумя критериями — скоростью выполнения и размером кода программы. Приходится жертвовать одним из них во имя другого, что сделать бывает довольно сложно, если конечный программный продукт должен в максимальной степени удовлетворять обоим из них.

Передо мной эта проблема возникла при разработке графического пакета, который должен был иметь и высокое быстродействие, и малый размер исполняемого кода (главным образом, при реализации алгоритмов построения примитивов и перемещения областей экрана). Одним из путей достижения приемлемого решения был выбран путь динамического изменения исполняемого кода программы, в зависимости от определенных реализуемым алгоритмом условий. То есть до определенного момента программа работает как обычно, потом в каком-то месте она изменяет свой собственный код, находящийся в ОЗУ, на нужный ей в данный момент при существующих условиях, и выполняет его. Особенно эффективно этот прием работает при динамическом изменении кода в теле цикла. Здесь мы получаем значительный выигрыш во времени без увеличения кода самой программы.

Для иллюстрации этого метода приводится реализация алгоритма перемещения и копирования областей экрана для адаптера EGA. На мой взгляд, этот пример наиболее ярко показывает достоинства динамической генерации кода программы при реализации некоторых алгоритмов. Текст этой подпрограммы написан на языке С и приводится в листинге. Компилятор — Borland C++ 2.0. Широко использовались ассемблерные включения, без которых реализация данного метода была бы очень затруднительна и почти невозможна. Следует заметить, что компиляция проводилась из командной строки через внешний ассемблер (опция B компилятора командной строки bcc). Использование встроенного ассемблера приводит к выдаче ошибок, таких как "неизвестная переменная", "неизвестная метка".

Динамическое изменение кода позволило уменьшить размер этой подпрограммы почти в 4 раза. Но при реализации этого метода существуют свои "подводные камни". Дело в том, что если мы попытаемся изменить команду, которая уже помещена в регистр очереди команд микропроцессора, то выполнится неизмененная команда, хотя в памяти она будет модифицированна. Поэтому изменению должны подвергаться команды, не помещенные еще в регистр очереди команд, то есть между командой изменения кода и самим изменяемым кодом должны быть несколько команд. В некоторых случаях можно использовать ничего не значащие "пустые" команды. Кроме команды mov, для модификации кода можно использовать также and, or, xor, которые выполняются несколько быстрее. В любом случае для получения машинного кода той или иной команды удобно пользоваться отладчиком debug, а далее выбирать, какой командой удобнее всего можно его сгенерировать или изменить.

С.Кириллов ■

Листинг

```
/————————Функция копирования (перемещения) областей экрана————————/
/*************************************************************/
/*Перемещение и копирование областей экрана в пределах одной или   */
/*двух страниц адаптера EGA с возможным их наложением друг на друга.*/
/*при расположении области-источника и области-приемника точно друг */
/*над другом на разных страницах осуществляется их своппинг (обмен). */
/*                                                           */
/*                                                           */
/*Автор: Кириллов С.А.                                       */
/*                                                           */
/*IBM PC AT/XT          23 сентября 1992 г.                  */
/*                                                           */
/*Компилятор: Borland C++ 2.0                                */
/*bcc -G -Z -O -C -B -a                                      */
/*                                                           */
/*                                                           */
/*Глобальные переменные: COPY = 1, MOVE = 0,                 */
/*                       Xmin,Ymin,Xmax,Ymax - границы экрана */
/*                                                           */
/*Параметры функции:                                         */
/*        what = COPY, MOVE    - определяет, перемещать или копировать*/
/*                               область;                    */
/*        from,to = 1,2        - определяют, с какой страницы на какую */
/*                               осуществлять перенос или копирование; */
/*        x1,y1 и x2,y2        - координаты левых верхних углов облас- */
/*                               ти-источника и области-приемника; */
/*                    w        - ширина области в пикселах;    */
/*                    h        - высота области в пикселах;    */
/*                                                           */
/*ВНИМАНИЕ! Все координаты должны быть выровнены на границу байта */
/*                                                           */
/*Возвращаемое значение:  1 - процесс прошел удачно,         */
/*                        0 - процесс осуществить невозможно */
/*************************************************************/
typedef unsigned char Byte;

int MoveWindow    (int what,int from,int to,int x1,int y1,int x2,int y2,
                   int w,int h)
{
    if(x2+w > Xmax || y2+h > Ymax || x2 < Xmin || y2 < Ymin) return 0;

    int offset_x = x2-x1;          // приращение по X;
    int offset_y = y2-y1;          // приращение по Y;
    int f = 32768;                 // смещение второй страницы;

    /* тест на взаимное расположение источника и приемника */
    if(offset_x = = 0 && offset_y = = 0)                {
        if(from ! = to && what = = MOVE)      {
            SwapWindow(x1,y1,w,h);
            return 1;                        }
        else if (what = = COPY && from = = to)  return 0;
    if (what = = MOVE && from = = to)           return 0;
                                                        }

condition(5,1);                    // установка режима копирования

asm {    xor  di,di                // очистка регистров;
         xor  si,si
    }

if(offset_y < 0)                   /* копирование сверху вниз */
    asm {    mov byte ptr cs:t_b3[1],0c7h     /* Изменение */  // add di,80
             mov byte ptr cs:t_b4[1],0c6h     /*    кода   */  // add si,80
        }
else {                             /* копирование снизу вверх */
    asm {    mov byte ptr cs:t_b3[1],0efh     /* Изменение */  // sub di,80
             mov byte ptr cs:t_b4[1],0eeh     /*    кода   */  // sub si,80
        }

    _DI = 80*h; _SI = _DI;   // заносим в DI и SI приращение в байтах по Y;
    }

_SI += Bx(x1,y1);  // добавляем смещение левого верхнего угла источника
_DI += Bx(x2,y2);  // добавляем смещение левого верхнего угла приемника

if(from = = 2) {    asm add  si,8000h // если источник на второй странице;
                    f = -32768; }
if(to    = = 2)     asm add  di,8000h // если приемник на второй странице;

if(what = = COPY) f = 0;           // в случае операции копирования;

asm {    mov ax,0a000h             // помещаем сегмент
         mov es,ax                    // видеопамяти EGA в ES;
         mov ax,w                  // делим ширину области
         shr  ax,1                 // в байтах на 8 и
         shr  ax,1                 // заносим результат в CX;
         shr  ax,1
         mov cx,ax
         mov ax,h
         inc  ax                   // заносим высоту области +1
         mov dx,ax                 // в DX;
    }

if(offset_x = 0)                   /* копирование справа налево */
    asm {    mov byte ptr cs:l_r1,4fh         /*          */  // dec di
             mov byte ptr cs:l_r2,4eh         /*Изменение */  // dec si
             mov word ptr cs:t_b1,0cf01h      /*   кода   */  // add di,cx
             mov word ptr cs:t_b2,0ce01h      /*          */  // add si,cx
             add  di,cx
             add  si,cx
             dec  di
             dec  si
        }
else                               /* копирование слева направо */
    asm {    mov byte ptr cs:l_r1,47h         /*          */  // inc di
             mov byte ptr cs:l_r2,46h         /*Изменение */  // inc si
             mov word ptr cs:t_b1,0cf29h      /*   кода   */  // sub di,cx
             mov word ptr cs:t_b2,0ce29h      /*          */  // sub si,cx
        }

asm mov bx,f

while(_DX—)  {    asm push cx
         cont: asm {    mov al,es:[si]          // считываем с источника;
                        mov byte ptr es:[di],0ffh // записываем в приемник;
                        mov al,es:[si+bx]       // считываем под источником;
                        mov byte ptr es:[si],0ffh // записываем в то же место,
                        l_r1:    dec di         // но в другой экран;
                        l_r2:    dec si
                        loop short cont
                        pop  cx
                        t_b1:    add di,cx      /* Изменяемые команды */
                        t_b2:    add si,cx
                        t_b3:    sub di,80
                        t_b4:    sub si,80
                    }
        }
condition(5,0);                    // восстановление режима записи 0;
return 1;
}       /* MoveWindow() */

/************ функция установки режима адаптера *************/
void condition (Byte p1, Byte p2)
{
    asm {    mov al,p1
             mov ah,p2
             mov dx,3ceh
             out  dx,ax
        }
}
/*********** функция вычисления адреса в видеопамяти ************/
unsigned int Bx (int x, int y)
{
    asm {    mov ax,y              // Умножаем строку на 80
             mov dx,ax             // и прибавляем столбец,
             mov bx,x              // разделенный на 8:
             mov cl,2              // y * 80 + x / 8
             shl  ax,cl            // ax := ax * 4
             add  ax,dx            // ax := ax * 5
             shl  cl,1             // cl := 4
             shl  ax,cl
             dec  cl               // cl := 3
             shr  bx,cl
             add  ax,bx
        }
    return (_AX);
}
/******** функция обмена областей друг с другом (свопшинг) *********/
void SwapWindow (int x, int y, int w, int h)
{
    condition(5,1); h++;
    _SI = Bx(x,y);
        asm {    mov di,si
                 add  di,8000h         // добавляем смещение второй страницы
                 mov ax,0a000h         // заносим сегмент видеопамяти
                 mov es,ax             // в ES;
                 mov ax,w              // вычисляем переменные цикла;
                 shr  ax,1
                 shr  ax,1
                 shr  ax,1
                 mov cx,ax
                 mov dx,28001          // смещение промежуточного байта;
            }
    begin:       asm push cx
cont:       asm {    mov al,es:[di]        // считываем со второго экрана;
                     mov bx,dx
                     mov byte ptr es:[bx],0ffh // записываем в промежуточный бай
                     mov al,es:[si]        // считываем с первого экрана;
                     mov byte ptr es:[di],0ffh // записываем во второй;
                     mov bx,dx
                     mov bx,dx
                     mov al,es:[bx]        // считываем промежуточный байт
                     mov byte ptr es:[si],0ffh // записываем в первый экран;
                     inc  di
                     inc  si
                     loop short cont
                     pop  cx
                     sub  di,cx
                     add  di,80
                     sub  si,cx
                     add  si,80
                     mov ax,h
                     dec  ax
                     mov h,ax
                     jne  short begin
                }
    condition(5,0);                    // восстанавливаем режим записи 0;
}       /* SwapWindow() */
```

Объектно-ориентированное программирование на BORLAND C++: контроль распределения памяти при отладке

А. Кельманзон

Моему другу и коллеге Михаилу Данилину посвящается

Введение

Интегрированная среда BORLAND C++ имеет хорошо развитую систему отладки, но подчас ее возможностей бывает явно недостаточно. Это связано с необходимостью проектирования сложных структур данных, где часто отследить правильность освобождения выделяемых блоков памяти весьма затруднительно. Как пример такого случая предлагается программа, позволяющая организовывать массивы строк разной длины (листинг 1).

Благодаря тому что класс VStrings, так же как и Strings, унаследован от класса Abstr, в векторы, организуемые объектами класса VStrings, могут помещаться сами объекты этого класса. В этом случае для того чтобы добиться правильного освобождения памяти, необходимо описать деструктор базового класса как virtual. Предположим, Вы забыли сделать это и Вам кажется, что программа написана правильно, потому что до какого-то момента это не будет отражаться на ее работе, и когда она "сломается", ошибку будет найти крайне сложно. Именно для обнаружения таких ошибок мной и была разработана программа CMEM.CPP.

Описание методов распределения и освобождения памяти в BORLAND C++

В рамках синтаксиса языка C++ все операции распределения и удаления блоков памяти осуществляются операторами NEW и DELETE. Для большей ясности приводятся исходные тексты реализации этих операторов (пример 1).

К сожалению, во встроенных подсказках нет информации о функции _new_handler (), а в документации по использованию этой функции посвящен всего один абзац в "Руководстве программиста".

Дело в том, что BORLAND C++ дает пользователю возможность самому обрабатывать ошибки распределения памяти (первоначально значение функции _new_handler равно нулю).

Существует функция:

```
pvf set_new_handler(pvf nf) {
    pvf of;
    of = _new_handler;
    _new_handler = nf;
    return of;
}
```

где pvf: typedef void (*_FAR pvf) (), которая позволяет изменять значение функции обработки ошибок распределения памяти.

Описание программы контролирования памяти CMEM.CPP

Основная идея основана на перегрузке глобальных операторов NEW и DELETE. В результате этого становится доступной информация о размере распреде-

Пример 1

```
void _FAR *operator new(size_t s) {
    void _FAR *p;
    s = s ? s : 1;
    while((p = malloc(s)) == NULL && _new_handler != NULL)
        _new_handler ();
    return p;
}
void operator delete(void _FAR *p) {
    if(p)
        free(p);
}
```

■

ляемых блоков, их адресах и памяти до их распределения и после их освобождения, а также информация о последовательности распределения и вложенности блоков.

При каждом вызове оператора NEW информация о распределяемом блоке заносится в массив структур HBL[] (информация о блоках). При вызове DELETE в соответствующий элемент заносится информация о количестве памяти после освобождения текущего блока. Все эти изменения отображаются в файле отчета <ИМЯ_EXE_ФАЙЛА>.RPT.

Для того чтобы не мешать работе исследуемой программы, в модуле CMEM.CPP отсутствует динамическое распределение памяти и, следовательно, есть ограничение на количество одновременно открытых блоков, но его легко изменить: достаточно в опциях компилятора в разделе DEFINE поставить объявление MAX_POINTERS = (возможное количество открытых блоков).

Кроме этого, в конце отчета приводится состояние HEAP после завершения программы.

Для использования модуля CMEM достаточно его добавить в проект. Текст программы CMEM.CPP приведен в листинге 2.

Пример получаемого отчета приведен на рис. 1.

```
Control Memory Allocation  Version 2.01 Copyright(c) 1991-92
       by Andrey K.Kelmanzon, ICS(55)
-------------------------------------------------------------

 [431536]
36A4:0004{
          Size: 100(112)
 [431424]
36AB:0004{
          Size: 200(200)
 [431216]
36B8:0004{
          Size: 300(304)
 [430800]
36D2:0004{
          Size: 400(416)
<36CB:0004>{?}
}[430384]
}[430912]
36CB:0004{
          Size: 500(512)
}[430912]
}[430912]
}[431536]

Heap:
```

#	Size	Status
1	528	used
2	528	used

```
Heap is OK.
```

Рис. 1. Пример получаемого отчета управления памятью в Вашей программе

В отчете приняты следующие обозначения.

– При описании блока распределяемой памяти:

```
[Память_до_выделения_блока]
Адрес_распределяемого_блока{
    Size: Количество_запрошенной_памяти
(Реально_распределенная_память)
}[Память_после_освобождения_блока]
```

– Когда для распределения блока используются функции malloc(), calloc() и тому подобные, а для удаления оператор delete:

```
<Адрес_блока>{?}
```

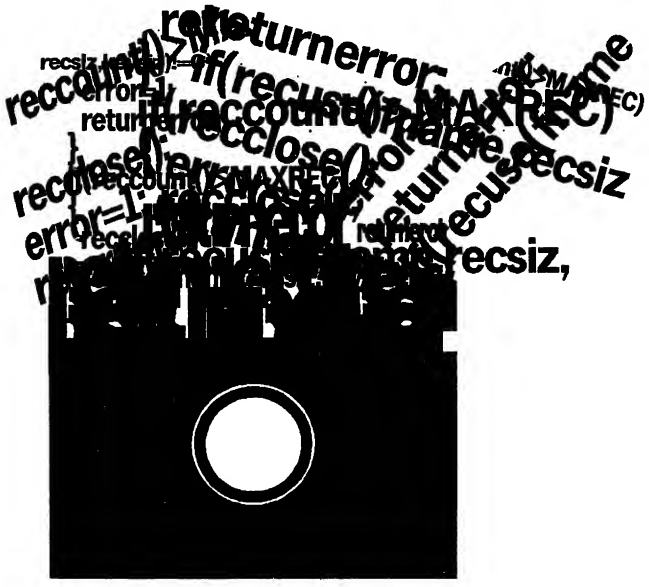
Листинг 1. Пример организации структуры данных, позволяющей обрабатывать массивы строк разной длины

```
#include <stdio.h>
#include <string.h>
class Abstr {          // Описание интерфейса работы
                       // с данными, помещенными в VStrings
    public:
                       // Заполнение элемента массива
        virtual  void      Get()          = 0;
                       // Создание нового элемента в массиве
        virtual  void      Put( Abstr* )  = 0;
                       // Печать данных, содержащихся в элементе
        virtual  void      Print()        = 0;
                       // ВИРТУАЛЬНЫЙ ДЕСТРУКТОР (!)
        virtual            ~ Abstr() {}
};
class Strings : public Abstr {  // Класс, описывающий строку
                                // произвольной длины
        char    *s;             // Содержимое строки
        int     size;           // Длина строки
    public:
                       // Конструктор
                       Strings( int i ){
                                s = new char [i];
                                size = i;
                       };
                       // Конструктор по умолчанию
                       Strings() { s = NULL; size = 0; };
                       // Деструктор
                     ~ Strings();
                       // Оператор доступа к каждому элементу строки
        char operator []     ( int i ) { return s[i]; };
                       // Пустое переопределение чистой
                       // виртуальной функции базового класса Abstr
        void Put( Abstr* ) {}
                       // Печать строки
        void Print() { printf("%s", s ); }
                       // Ввод строки
        void Get(){
                                char gg[100];
                                scanf( "%s", gg );
                                strcpy( s , gg );
                       }
};
Strings::~Strings( void ){
        delete s; s = NULL;
        size = 0;
}
class VStrings : public Abstr { // Класс, описывающий вектор классов,
                                // порожденных от Abstr
        Abstr * *d;             // Вектор указателей на классы
        int     cur,            // Текущий элемент вектора
                size;           // Размер вектора
    public:
                       // Конструктор по умолчанию
                       VStrings(): d(NULL), cur(0), size(0) {}
                       // Конструктор
                       VStrings( int i ){
                                d = new Abstr*[i];
                                cur = 0; size = i;
                                }
                       // Деструктор
                       ~VStrings();
                       // Оператор доступа к каждому элементу вектора
        Abstr     &operator [](int i) { return *d[cur=i]; };
                       // Помещение объекта в текущий элемент вектора
        void       Put( Abstr* z ) { d[ cur++ ] = z; };
                       // Печать вектора
        void       Print();
                       // Ввод вектора
        void       Get();
};
    VStrings::~VStrings() {
    for( int i = 0; i<size; i++ )
        delete d[i];
    delete d;
}
void VStrings::Print() {
        for( int i = 0; i <size; i++ )
                d[i]->Print();
}
void VStrings::Get() {
        for( int i = 0; i < size; i++ )
                d[i]->Get();
}
#define COL 2
void main ( void )
{
VStrings Vol(COL);
    for( int i = 0; i < COL; i++ ) {
    // Размещение в каждый элемент вектора Vol по вектору размерности COL
        Vol.Put(  new VStrings( COL ) );
    }
    for( i = 0; i < COL; i++ )
        for( int j = 0; j < COL; j++ ) {
    // Все строки, находящиеся в массиве, будут иметь размерность 20 символов
            Vol[i].Put( new Strings( 20 ) );
        }
    // Заполнение информацией созданной структуры данных
    for( i = 0; i < COL; i++ )
        for( int j = 0; j < COL; j++ )
            ((VStrings&)Vol[i])[j].Get();
    // Печать содержимого Vol
    for( i = 0; i < COL; i++ )
        for( int j = 0; j < COL; j++ )
            ((VStrings&)Vol[i])[j].Print();
}
```

Листинг 2. Исходный текст программы контролирования памяти CMEM.CPP

```
//-------------------------------------------------------------
//
//          CMEM,   v. 2.01
//       КОНТРОЛЬ РАСПРЕДЕЛЕНИЯ
//       ПАМЯТИ ДЛЯ РАЗРАБОТКИ
//       ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ
//       ПРИЛОЖЕНИЙ
//
//  Институт Проблем Управления ( л.55 )
//  Андрей К.Кельманзон
//
//  Использование:
//  ---------
//     - скопируйте в рабочую директорию файл CMEM.CPP;
//     - добавьте в проект файл CMEM.CPP;
//     - по умолчанию программа работает со 100
//       одновременно распределенными указателями,
//       если Вам необходимо изменить это значение
//       Options-> Compiler-> Code generation-> Defines
//       MAX_POINTERS = ##;
//     - после окончания работы вашей программы
//       на диске будет создан файл CMEM.RPT,
//       который состоит из "дерева" распределения
//       памяти.
//-------------------------------------------------------------
#include <alloc.h>
#include <new.h>
#include <mem.h>
#include <stdlib.h>
#include <dos.h>
#include <stdio.h>
#include <process.h>

#if !defined(MAX_POINTERS)
#define MAX_POINTERS 500      // Максимальное число вызовов NEW
#endif

#define MEM_TO0( __nBL ) memset( __nBL,     0, sizeof __nBL )

extern unsigned _stklen = 43210U;

//Описатели ////////////////////////////////////////////
enum flag       { Off, On, End = -1 };
enum line_char { U='\xDA', D='\xC0', H='\xC4',
                 V='\xB3', C='\xC5', S=' ' };

typedef unsigned long ulong;
typedef unsigned long uint;
typedef void (_FAR *pvf)();

// Получает адрес текущего PSP ////////////////////////////
unsigned GetCurPSP( void ) {
     asm {
          mov AX, 0x6200
          int  0x21
     }
     return _BX;
}

// Формирует имя файла отчета /////////////////////////////
// EXE_FILE_NAME.rpt
//-------------------------------------------------------
void GetNameEXE( char name[] ) {
     char      far  *ptr;
     char      *ext = ".rpt";
     int                          i = 0;
          // Pointer on ENVIROMENT
          ptr = (char far*)
                    // Addres of ENVIROMENT
                    ( ( long )(*(int far*)
                    // Addres of pointer to current DOS ENVIROMENT
                    (((long )GetCurPSP() < <16) + 0x2C)
                    ) < <16 );
          while( *ptr++ != '\x1' ); ptr += 1;
          while( *ptr != '.' )
                    name[i++] = *ptr++;
          for( int j = 0; j < 4; j++ )
                    name[i++] = ext[j];
}

// Описание блока памяти //////////////////////////////////
struct HeapBlock {
     void _FAR     * Addr;          // Адрес
     ulong         SizeHOST,        // Размер:    - запрошенный
                   SizeALLOC;       //            - распределенный
     flag          Status;          // Статус занятости
     int           NCall;           // Номер в time_call
     // Количество памяти ----------------------------------
     ulong         MemBefore,       // До распределения
                   MemLater;        // После освобождения
};

// Класс, контролирующий и обрабатывающий ошибки распределения памяти
class ControlMem {
public:
     ControlMem( void );                // Конструктор
     ~ControlMem( void );               // Деструктор

     int   search( void _FAR * );       // Поиск адреса
     int   search_free( void );         // Поиск свободного блока
     // Перегруженные глобальные операторы распределения памяти ---
     friend void _FAR * operator new( size_t );
     friend void        operator delete( void _FAR *);

private:
     struct HeapBlock   HBL[ MAX_POINTERS ]; // Массив описателей блоков
     static flag        StatusClass;    // Статус присутствия класса
                                        // On-да Off-нет
     static flag        CloseBlock;     // Статус "закрыт ли предыдущий
                                        // блок" On-да Off-нет
     FILE               *rpt;           // Указатель потока файла отчета
     static int         Meter;          // Счетчик вызовов new & delete
     line_char          ch;  // Символ линии

     flag  time_call[ MAX_POINTERS*2 ]; // Порядок вызовов
                                        // операций с блоками
     int   find_state( void );          // Возвращает номер
                                        // в time_call[]
     void state_end( void );            // Выставляет признак конца
     // Обработка ошибок распределения памяти ----------------
     pvf   old_handler;  // Указатель на старую функцию
                         // обработки ошибок распределения
     // Функции формирования отчетов -------------------------
     void open_r( void );               // Открытие
     void close_r( void );              // Закрытие
     void out_mem  ( int );             // Размер максимального блока
     void out_size ( int );             // Размер выделенного блока
     void out_addr ( int );             // Адрес блока
     void out_tab  ( int = -1 );        // Отступ
     void out_space( void );            // Пробел
     void out_open ( int );             // Открытие блока
     void out_close( int );             // Закрытие блока
     void out_unknown( void _FAR* );    // Неизвестный блок
     void out_heap( void );             // Выводит состояние динамически
                                        // распределяемой области памяти
};

// Инициализация static ///////////////////////////////////
flag ControlMem::StatusClass = Off;     // Класс обработки отсутствует
flag ControlMem::CloseBlock  = Off;     // Нет закрытых блоков
int  ControlMem::Meter       = 0;       // Счетчик вызовов

// Объявление класса обработки ////////////////////////////
ControlMem MemAlloc;

// Конструктор класса обработки ///////////////////////////
ControlMem::ControlMem( void ) {
     ControlMem::StatusClass = On;
     MEM_TO0( HBL );
     MEM_TO0( time_call );
          time_call[0] = End;       // End - Признак конца
     open_r();
}

// Открытие файла отчета //////////////////////////////////
// Принцип работы:
//   в случае невозможности создания файла - stderr
//-------------------------------------------------------
void ControlMem::open_r( void ) {
     char n[129] = "";
     GetNameEXE( n );
     if( (rpt = fopen( n, "wt" )) == NULL )
          rpt = stderr;
     fprintf( rpt,
     "-------------------------------------------------------\n"
     "Control Memory Allocation   Version 2.01 Copyright(c) 1991-92   \n"
     "            by Andrey K. Kelmanzon, ICS(55)                    \n"
     "-------------------------------------------------------\n\n"
     );
}

// Закрытие файла отчета //////////////////////////////////
void ControlMem::close_r( void ) {
     if (rpt != stderr)
          fclose( rpt );
}

// Размер максимального свободного блока //////////////////
void ControlMem::out_mem( int n ) {
     switch( HBL[ n ].Status ){
          case On:
                    fprintf( rpt, "[%lu]\n", ( ulong )HBL[ n ].MemBefore );
                    break;
          case Off:
                    fprintf( rpt, "[%lu]\n", ( ulong)HBL[ n ].MemLater );
     }
}

// Запрашиваемый и реально распределенный размеры блока "n"
//////
void ControlMem::out_size( int n ) {
     fprintf( rpt, "              Size: %lu(%lu)\n",
```

```
            ( ulong )HBL[ n ].SizeHOST,
            ( ulong )HBL[ n ].SizeALLOC
        );
}

// Адрес блока
//////////////////////////////////////////////
void ControlMem::out_addr( int n ) {
        fprintf( rpt, "%c%Fp{\n", ch = U, HBL[ n ].Addr );
}

// Отступ состоит из вертикальных линий в соответствии с time_call //
void ControlMem::out_tab( int d ) {
        int i = 0;
        if(d != -1) d = HBL[ d ].NCall;
        while( time_call[ i ] != End ) {
                if (d == -1 || i<d)
                        if( time_call[ i ] ) ch = V;
                        else ch = S;
                else if( i == d ) ch = D;
                else if( i > d )
                        if( time_call[ i ] ) ch = C;
                        else ch = H;
                putc( ch, rpt ); i++;
        }
}

// Печатает пробел ////////////////////////////////////////
void ControlMem::out_space( void ) {
        putc( '\x20', rpt );
}

// Печатает информацию об открывающемся блоке ////////////////
void ControlMem::out_open( int n ) {
        if( !ControlMem::CloseBlock ) {
                out_tab(); out_space(); out_mem( n );
        }
        out_tab(); out_addr( n );
        HBL[ n ].NCall = MemAlloc.find_state();
        time_call[ HBL[ n ].NCall ] = On;
        out_tab(); out_size( n );
}

// Печатает адрес закрывающегося блока, который не открывался ////
void ControlMem::out_unknown( void _FAR *p ) {
        out_tab(); fprintf( rpt, "<%Fp> {?}\n", p );
}

// Закрывает блок "n" ////////////////////////////////////
void ControlMem::out_close( int n ) {
        out_tab( n ); putc( '}', rpt ); out_mem( n );
        time_call[ HBL[ n ].NCall ] = Off;
        state_end();
}

// Информация о динамически распределяемой области памяти /////
void ControlMem::out_heap( void ) {
        heapinfo hi;
        int      i = 0;
        fprintf( rpt, "\nHeap:\n" );
        fprintf( rpt, "-----------------------\n");
        fprintf( rpt, "| # | Size | Status |\n" );
        fprintf( rpt, "-----------------------\n" );
        hi.ptr = NULL;
        while( heapwalk( &hi ) == _HEAPOK )
                fprintf( rpt, "| %3d | %7lu | %4s |\n",
                        ++i,
                        (ulong)hi.size,
                        hi.in_use ? "used" : "free"
                );
        fprintf( rpt, "-----------------------\n" );
        if(heapcheck() == _HEAPCORRUPT)
                fprintf( rpt, "\nHeap is corrupted.\n" );
        else
                fprintf( rpt, "\nHeap is OK.\n" );
}

// Деструктор класса обработки ////////////////////////////
ControlMem::~ControlMem( void ) {
        ControlMem::StatusClass = Off;
        out_heap();
        close_r();
        //   set_new_handler( old_handler );
}

// Возвращает номер в time_call[] блока i ////////////////////
int ControlMem::find_state( void ) {
        int k = 0, i = 0;
        while( time_call[i] != End )
                if( time_call[i++] == On ) k = i;
        time_call[ k ] = Off; time_call[ k+1 ] = End;
        return k;
}

// Выставляет признак конца ////////////////////////////////
void ControlMem::state_end( void ) {
        int i = 0, k;
        while( time_call[i++] != End );
        k = (--i);
        while( i >= 0 ) {
                switch( time_call[ i ] ) {
                        case On: return;
                        case Off: time_call[ k ] = Off;
                                  time_call[ i ] = End;
                                  k = i;
                }
                i--;
        }
}

// Поиск информации о блоке ////////////////////////////////
// Передается:
//   p-адрес блока
// Возвращает:
//   номер блока (i) или -1, если он не найден
//-----------------------------------------------------
int ControlMem::search( void _FAR *p ) {
        for( int i = 0; i < MAX_POINTERS; i++ )
                if (HBL[ i ].Addr == p)
                        return i;
        return -1;
}

// Поиск свободного описателя //////////////////////////////
int ControlMem::search_free( void ) {
        int i = 0;
        do{
                if (HBL[ i ].Status == Off)
                        return i;
        }while(++i < MAX_POINTERS);
        return -1;
}

// Исходный оператор new ///////////////////////////////////
void _FAR *o_NEW( size_t s ) {
        void _FAR *p;
        s = s ? abs( s ) : 1;
        while ( (p = malloc( s )) == NULL && _new_handler != NULL )
                _new_handler();
        return p;
}

// Перегруженный оператор new //////////////////////////////
// Принцип работы:
//   если класс обработки отсутствует, работает как обыкновенный new
//-----------------------------------------------------
void _FAR *operator new( size_t size_block ) {
        int              i;
        void _FAR     * ptr;
        ulong            mb = coreleft();
        ptr = o_NEW( size_block );
        if( ControlMem::StatusClass ){
                if( (i = MemAlloc.search_free()) == -1 ){
                        ControlMem::StatusClass = Off;
                        MemAlloc.out_heap();
                        MemAlloc.close_r();
                        fprintf(stderr, "\nStack of pointers is full.\n");
                        abort(); //exit(0);
                }
                MemAlloc.HBL[ i ].SizeHOST = ( ulong )size_block;
                MemAlloc.HBL[ i ].SizeALLOC = mb - coreleft();
                MemAlloc.HBL[ i ].Status    = On;
                MemAlloc.HBL[ i ].MemBefore = mb;
                MemAlloc.HBL[ i ].Addr      = ptr;
                MemAlloc.out_open( i );
        }
        ControlMem::CloseBlock = Off;
        return ptr;
}

// Исходный оператор delete ////////////////////////////////
void o_DELETE( void _FAR *ptr ) {
        if( ptr ) free( ptr );
}

// Перегруженный оператор delete ///////////////////////////
// Принцип работы:
//   если класс обработки отсутствует, работает как обыкновенный delete
//-----------------------------------------------------
void operator delete( void _FAR *ptr ) {
        int      i;
        o_DELETE( ptr );
        if( ControlMem::StatusClass ){
                if( (i = MemAlloc.search( ptr )) == -1 ){
                        MemAlloc.out_unknown( ptr ); return;
                }else{
                        MemAlloc.HBL[ i ].MemLater = coreleft();
                        MemAlloc.HBL[ i ].Status = Off;
                        MemAlloc.out_close( i );
                }
        }
        ControlMem::CloseBlock = On;
}
```



Пишем упаковщик

Д. Арапов

Введение

Большинству программистов известны программы-архиваторы: PKPAK, PKARC, PKZIP, LHARC, ARJ и т.п. Они сжимают хранящуюся в файлах информацию и помещают ее в файл-архив, где она хранится до тех пор, пока не понадобится. Восстановить первоначальный вид данных можно, воспользовавшись процедурой распаковки. Изучению сравнительных достоинств архиваторов посвящено множество статей. Напомним, что для оценки архиваторов обычно используют два критерия: коэффициент сжатия и скорости упаковки и распаковки. Эти характеристики обычно зависят от характера информации: исполняемые программы, графика, текст, записи базы данных и так далее.

Цель данной статьи - познакомить читателя с идеями, положенными в основу современных упаковщиков, и предложить программу компрессии-декомпрессии, представляющую собой вариацию на тему алгоритма Лемпела-Зива. Этот алгоритм позволяет паковать данные любых типов, однако прежде всего он ориентирован на сжатие текстовой информации. Приведенный модуль читатель может поместить в собственный набор инструментов и использовать в своих программах (скажем, в инсталляторах).

Выделятюся два типа организации алгоритмов упаковки: одно- и двухпроходные. И те и другие используют словарь, в котором накапливают статистику о повторяющихся символах и подстроках. В случае двухпроходной организации на первом проходе строится словарь, а на втором (при помощи этого словаря) производится упаковка. Как правило, двухпроходные алгоритмы позволяют достичь чуть более высокой степени сжатия, однако организация дополнительного прохода требует больше времени, кроме того, чтобы обеспечить распаковку, приходится запоминать вместе с текстом и словарь.

Алгоритм Лемпела-Зива

Алгоритм Лемпела-Зива, которому посвящена статья, имеет однопроходную организацию. Как в процессе упаковки, так и при распаковке создается словарь, так что сохранять его вместе с текстом не нужно. В словарь заносятся подстроки исходного текста. При упаковке они заменяются номерами, под которыми они найдены в словаре. Опуская детали, алгоритм сжатия можно представить на псевдокоде таким образом:

```
Инициализировать словарь.
Установить текущую позицию в начало
текста.
ПОКА не весь текст запакован:
    Выделить наиболее длинную, начинающуюся в текущей позиции подстроку s,
    для которой есть соответствие в словаре.
    Записать в архив номер позиции P найденной подстроки в словаре.
    Посмотреть следующий за подстрокой
    в тексте символ K.
    Занести в словарь строку sK.
    Продвинуть текущую позицию, чтобы
    она указывала на символ K.
КОНЕЦ ПОКА
```

Алгоритм распаковки выглядит так:

```
Инициализировать словарь.
Установить позицию на начало архива.
ПОКА не весь архив распакован:
    Прочесть из архива очередное число P.
    Найти строку s, лежащую на позиции P
    в словаре. Записать s в конец распакованного текста. Прочесть из архива
    следующее за P число Q и найти в словаре строку u, находящуюся на месте Q.
    Записать в словарь su[1]. Продвинуть
    текущую позицию на символ Q.
КОНЕЦ ПОКА
```

Принцип одинакового изменения словаря

Основной принцип, на основе которого построены все рассматриваемые вариации алгоритма, состоит в том, что упаковщик и распаковщик изменяют словарь одинаковым образом. Благодаря этому получаемая при декомпрессии из словаря по номеру n строка совпадает со строкой, которую упаковщик зашифровал тем же числом n.

Отметим, что приведенный выше алгоритм не вполне точен. Проблема состоит в том, что если упаковщик может немедленно занести расширенную строку в словарь, то при распаковке этого сделать нельзя, необходимо прочесть следующую строку. Поэтому для обеспечения одинаковой последовательности манипуляций со словарем при упаковке расширенную строку помещают для начала во вспомогательный буфер, а в словарь заносят только на следующем витке цикла.

Строки вида KsKsK

В большинстве случаев этот прием никак не влияет на качество сжатия. Однако это не так, если в пакуемом тексте, начиная с текущей позиции, находится строка вида KsKsK, где K - некоторый символ, а s - подстрока (возможно пустая). Пусть подстрока Ks ранее была занесена в словарь и ей был присвоен некоторый номер x, в то же время строка KsK в словаре отсутствует. Если следовать исправленному алгоритму, то строка KsKsK будет кодироваться как xxy, где y - код подстроки, начинающейся с третьего вхождения символа K. Модифицируем алгоритм, введя дополнительную проверку, не находится ли подстрока в буфере. Подстроку KsK будем кодировать особым числом z, не являющимся правильным номером словаря. Это позволит закодировать исходную строку KsKsK в виде xz. При декомпрессии, обнаружив номер z, мы не должны обращаться к словарю. Вместо этого нам следует воспользоваться полученной на предыдущем витке цикла строкой.

Замечание. Вся приведенная выше "кухня" не заслуживала бы внимания, если бы не весьма распространенные строки вида 'K...KKK'. На них использование приведенного выше приема позволяет увеличить плотность кодирования до 2 раз.

Кодирование позиций

До сих пор мы не говорили о том, в каком виде номера входов словаря записываются в сжатом представлении (архиве). Если число входов в словаре меньше 2**n, то каждый вход можно закодировать в виде цепочки из n битов. Обратите внимание, что неравенство строгое, поскольку одну комбинацию битов нужно зарезервировать для описанного выше случая. Обычно используются словари размером от 2**10-1 до 2**14-1 входов. Однако тривиальная кодировка позиций не самая экономная. Гораздо лучше кодировать, используя цепочки битов переменной длины: короткие для часто встречающихся номеров позиций, а длинные для редко встречающихся. При этом, однако, потребуется по крайней мере один дополнительный бит для кодирования длины цепочки.

Организация словаря

Словарь должен позволять искать позицию по строке (при упаковке) и строку по позиции (для распаковки). Кроме этого, словарь должен быть организован так, чтобы позиция для наиболее употребимых в последнее время строк кодировалась с помощью меньшего числа знаков. Рассмотрим организацию словаря в виде приоритетного списка: в головной части (с номерами 1, 2 и так далее) будут помещаться наиболее употребительные подстроки, а относительно редкие строки попадут в хвост списка (tail). При этом для изменения словаря будут использоваться 3 операции:

Add (включить)	–	добавляет в словарь новую строку;
Delete (исключить)	–	удаляет из словаря строку;
Promote (продвинуть)	–	перемещает ранее помещенную в словарь строку ближе к началу списка.

Две операции - Add и Promote - вызываются непосредственно из процедур упаковки (распаковки), а Delete - изнутри операции Add в том случае, когда словарь уже целиком заполнен: она выбирает и удаляет из словаря наименее полезную, с ее точки зрения, строку. При этом следует соблюдать 2 принципа:

1. Нельзя исключать из словаря односимвольные строки.

2. Нельзя исключать из словаря строку, которая является префиксом другой строки словаря или добавляемой строки.

Нарушение первого принципа (инварианта) может остановить процесс сжатия: алгоритм Лемпела-Зива основан на том, что, пока текст полностью не упакован, мы можем отщепить от него непустую строку, которая уже присутствует в словаре. Для этого там должны быть как минимум все односимвольные строки. Нарушение второго принципа приведет к тому, что в словаре появятся недостижимые строки.

В рассматриваемой программе мы будем исключать первый с конца элемент словаря, не попадающий под перечисленные запреты.

Адаптивность

Операции Add и Promote могут быть закодированы по-разному. От выбранного способа зависит степень

адаптивности кодирования. Дело в том, что сжимаемые файлы могут быть неоднородны. Например, текст этой статьи практически не содержит строк из латинских букв, а включенный в нее пример программы - знаков кириллицы. Способность словаря быстро подстраиваться к изменениям частотных характеристик входного текста называется адаптивностью.

Если нужно внести в словарь новую строку, то ее можно добавить в хвост списка. Однако после того как словарь заполнен этот подход практически не позволяет внести в него новую строку, поскольку с очень большой вероятностью при вставке следующего элемента эта строка будет удалена, ведь она находится в самом конце списка. Таким образом в словаре возникнут "застойные явления" и он не будет адаптироваться к изменениям во входном тексте, в свою очередь, это приведет к тому, что соотвествующая программа сжатия будет плохо работать на длинных и неоднородных текстах.

Другой крайностью будет включение нового элемента в начало списка. Безусловно, этот способ очень адаптивен, однако в начало списка очень часто будут попадать случайные строки, оттесняя с выгодных позиций в середину списка часто повторяющиеся элементы. В результате последние будут кодироваться при помощи более длинных последовательностей битов и общее качество упаковки снизится.

Аналогичные рассуждения применимы и к операции Promote. Новые элементы предлагается включать в позицию Count div m, где Count - число элементов в списке, операция Promote будет каждый раз передвигать элемент из позиции n в позицию n div m. В результате экспериментов с различными текстами было обнаружено, что лучшие результаты обычно достигаются при m = 3.

Поиск в словаре

Исторически ахиллесовой пятой упаковщиков была низкая скорость поиска строки в словаре. Кроме этого, манипуляция со строками переменной длины обычно подразумевает использование динамически распределяемой области памяти (все равно, стандартной или реализованной самостоятельно) и, соответственно, операций new и dispose, требующих сравнительно много времени. Во многих упаковщиках последняя проблема решена довольно просто: фрагмент пакуемого текста (длиной обычно 4 или 8K), непосредственно предшествующий текущей позиции компрессора (декомпрессора) хранится в циклическом буфере. Есть и другой способ. Хранения строк переменной длины можно избежать, если учесть следующие особенности.

1. Если в словаре имеется некоторая строка, то он должен содержать и все ее префиксы. Как следствие, каждую строку длины l можно представить как пару из последнего символа и ссылки на префикс длиной l - 1.
2. Если на некотором шаге алгоритма упаковки производится поиск строки sK, то на предыдущем шаге был найден ее максимальный префикс - подстрока s.

В нашей программе для обеспечения приемлемой скорости используется структура данных, состоящая из записей фиксированной длины. При этом каждая запись содержит символьное поле и 4 ссылки на другие записи. Все записи помещены в массив, а ссылки реализуются с помощью индексов массива. Кроме этого, допустимым значением ссылки будет специальное значение NullLink (ссылка "в никуда"). В целом построенная структура данных будет иметь вид дерева.

```
type TNode  =  record
                  Up, Down
                  Left, Right   : TLink;
                     Symbol     : Char
               end;
```

Рассмотрим назначение полей. Ссылка Up указывает на узел, содержащий предыдущую литеру кодируемой строки (то есть на максимальный префикс строки). Таким образом записи типа TNode (узел) взаимно однозначно соответствует строка словаря, и мы можем далее не различать их. Для односимвольных строк ссылка Up будет указывать "в никуда". Left и Right ссылаются на узлы, представляющие строки той же длины, которые отличаются от данной только последним символом. Эти поля служат для организации бинарного поиска среди строк с одним и тем же максимальным префиксом. Ссылка Down будет указывать на одну из строк, для которых данная будет максимальным префиксом (рис. 1).

		Up	Down	Left	Right	Sym
1	a	2	5	0	0	a
2	b	0	0	1	3	b
3	c	2	0	0	4	c
4	d	3	0	0	0	d
5	ab	1	7	0	6	b
6	ad	5	0	0	0	d
7	abb	5	0	9	8	b
8	abc	7	0	0	0	c
9	aba	7	0	0	0	a

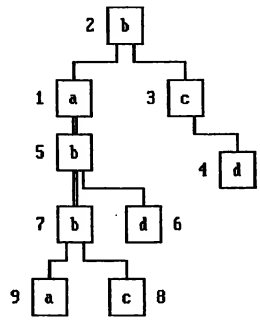


Рис. 1. Представление строк в словаре. Двойной линией обозначены ссылки Down

К сожалению, операции Add, Delete и Promote не ограничиваются изменением позиции одного элемента. Входы словаря постоянно меняют свои места. Корректировка ссылок может потребовать очень много времени. Для решения этой проблемы можно воспользоваться двумя способами.

1. Использовав представление словаря в виде двух массивов: массива записей типа Node (узлов дерева), в котором каждая запись от момента создания до мо-

мента уничтожения не меняет своего места, и массива ссылок на элементы этого массива. Операции Add, Delete и Promote изменяют взаимное расположение ссылок на элементы словаря, а не расположение самих элементов.

Дотошный читатель заметит подвох и спросит: "А как организован поиск позиции ссылки на строку в приоритетном списке?" Увы, линейным просмотром. Однако на IBM PC этот просмотр можно сделать весьма быстрым, запрограммировав его на ассемблере и воспользовавшись командой REP SCASW (поиск слова в массиве). К нашей выгоде действуют также два обстоятельства: во-первых, искомое значение заведомо присутствует в массиве, а во-вторых, в силу приоритетности списка, расположено скорее всего ближе к началу списка (рис. 2).

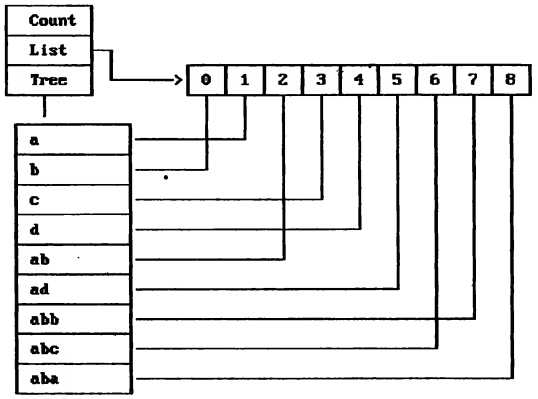


Рис. 2. Словарь представлен в виде массива Tree узлов дерева, на которые идут ссылки из приоритетного списка List

2. Имеется и другой более быстрый (по крайней мере, теоретически) алгоритм: заранее разделить список на две части (в принципе, можно и на большее число частей, но выигрыша в плотности упаковки это практически не дает): на маленькую и большую. И та и другая представляют собой двунаправленные списки элементов массива. Новые узлы включаются с головы списка в большую часть. Если какой-либо элемент из большой части встречается чаще, чем самый долго не встречавшийся из малой, то они обмениваются местами. Только в случае таких обменов приходится перестраивать ссылки Up, Down, Left и Right (рис. 3).

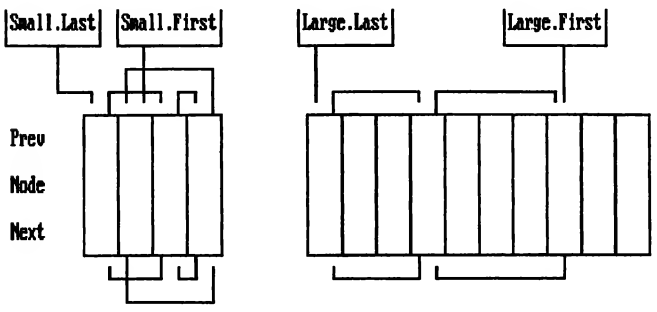


Рис.3. Представление словаря в виде двух частей: малой (Small) и большой (Large). Обратите внимание, что в части Large не все элементы включены в двунаправленный список

В приводимой программе малая часть состоит из 127 входов, а большая из 3966. Каждый элемент из малой части может быть представлен 7 битами, а из большой - 12, кроме этого, требуется по одному биту для того чтобы различать коды.

Мы уже заметили, что для каждой строки, которую мы ищем в словаре при упаковке, ее префикс уже найден в словаре. Благодаря рассмотренной организации словаря время поиска строки не зависит от количества входов в словарь.

Инициализация словаря

Первоначально в словарь заносятся 256 различных односимвольных строк. Если известно, что программа будет работать с особыми текстами (например, с программами на Паскале), то при инициализации словаря имеет смысл занести характерные для таких текстов строки, разумеется, со всеми их префиксами. Не забудьте, что инициализация словаря для программы сжатия и для распаковки должна производиться одинаково.

Конфликты

Алгоритм LZ всегда отщепляет от пакуемого текста начальную подстроку максимальной длины. В большинстве случаев это наиболее выгодная стратегия. Однако часто встречаются ситуации, когда, отщепив вначале более короткий префикс, можно выиграть. Это происходит в тех случаях, когда строка, идущая вслед за коротким префиксом, значительно длиннее, чем строка, начинающаяся вслед за префиксом максимальной длины, и сумма длин двух последовательных подстрок больше, чем сумма длин двух последовательных подстрок, выделяемых алгоритмом LZ. Назовем такую ситуацию конфликтом. Приведем практический пример возникновения конфликта: пакуется текст программы на Паскале. В словаре имеются все 256 символов, а также строки: 'p', 'pr', 'pro', 'proc'. Начиная с текущей позиции, текст имеет вид ' proc'. Если отщепить префикс максимальной длины (' p'), то этот фрагмент можно будет закодировать четырьмя входами словаря. Если же наш алгоритм умеет заглядывать вперед, то тот же фрагмент можно представить в виде всего двух строк словаря: ' ' + 'proc'.

Алгоритм с разрешением конфликтов выбирает тот из префиксов, для которого сумма его длины и длины самой большой начинающейся после него строки максимальна. Его следует рекомендовать только в том случае, когда плотность упаковки имеет решающее значение. Использование его замедляет работу в несколько раз (коэффициент замедления приблизительно равен отношению плотностей упакованного и исходного текстов, то есть 2-3).

Короткие тексты и энергичное пополнение словаря

Алгоритм Лемпела-Зива предназначен для упаковки сравнительно больших текстов, объем которых в

символах существенно больше числа входов словаря. Маленькие тексты он пакует хуже. Это связано с тем, что в начале работы словарь практически пуст. Кодирование одной строки расширяет словарь на 1 элемент. Если программу упаковки предполагается применять для работы с относительно короткими текстами, скажем, в информационно-поисковой системе, то имеет смысл более активно пополнять словарь. Предлагается добавлять в словарь строки вида s + t, где t - следующая подстрока текста, найденная в словаре. Разумеется, чтобы избежать противоречия принципу доступности, в словарь следует занести и все префиксы, которые длиннее s. Основным преимуществом будет быстрое попадание в словарь длинных, но часто повторяющихся подстрок.

Так, при кодировании программы на Паскале алгоритму Лемпела-Зива потребуется минимум 8 раз встретить слово 'function', чтобы закодировать его при помощи одного числа. Предложенный алгоритм способен сделать это уже с 4 попытки.

Недостаток этого подхода очевиден: быстрое разбухание словаря. У этого разбухания имеется 2 негативных последствия: во-первых, могут быть вытолкнуты более короткие и полезные подстроки, во-вторых, потребуется больше двоичных знаков для кодирования входов. Когда речь идет о коротких текстах, первым можно пренебречь: количество строк, попавших в словарь, не превысит числа символов в исходном тексте. Второе последствие также не столь существенно при использовании кодов переменной длины: активные входы будут по-прежнему заменяться короткими цепочками битов, а длина кодов для строк, которые были занесены в словарь, но больше не встретились, на объем упакованого текста не влияет.

Заключение

Представленная программа позволяет упаковывать текст приблизительно так, как это делали архиваторы первого поколения: PKPAK, ранние версии программы ZIP. Она может оказаться полезной для тех, кто по каким-либо причинам не пользутеся существующими архиваторами, скажем, при создании какой-нибудь своей программы. ■

Литература

1. Lempel A., Ziv J. A Universal Algorithm for Sequential Data Compression// IEEE Trans. Info. Theory.- 1977.- N 23.- P.337-343.
2. Lempel A., Ziv J. Compression of Individual Sequences Via Variable-Rate Coding// IEEE Trans. Info. Theory.- 1978.- N 24.- P.530-536.
3. Welsh T.A. A Technique for High-Performance Data Compression// Computer.- 1984.- N 17.- P.8-19.
4. Chang D.K. Data Compression Using Hierarchical Dictionaries// The Journal of Systems and Software.- 1991.- Vol. 15.- N 3. - P. 233-238.

Всем,

кто имеет дело с однокристальными микро-ЭВМ популярных серий **Intel 8051** и **Intel 8048** или их аналогами,

фирма "ПРОЛОГ-88" предлагает:

♦ *ВНУТРИСХЕМНЫЕ ЭМУЛЯТОРЫ*

- эмуляция ОЭВМ без ограничений *в реальном времени*
- неограниченное количество аппаратных *точек останова* по всему адресному пространству
- аппаратная *трассировка* в реальном времени по 36 каналам на глубину 2К кадров с заданием момента начала (конца) трассировки программно или по сочетанию внешних сигналов
- встроенный *программатор* ОЭВМ и ППЗУ
- возможность *автономной работы* на объекте с помощью пульта
- *гарантия* на все устройства 1 год
- *символьная отладка*
- поддержка языков *высокого уровня*
- удобный доступ к данным с помощью *ресурсных фреймов*, конструируемых пользователем

Дизайн наших устройств украсит Ваше рабочее место!

♦ *ПРОГРАММНЫЕ СИМУЛЯТОРЫ*

- полное моделирование *ОЭВМ* и *внешней памяти*
- имитация *ввода-вывода*
- имитация *прерываний*
- *единый интерфейс* с отладчиком эмулятора

Мы также можем разработать и наладить производство *контроллеров на базе ОЭВМ* по Вашему техническому заданию.
Высокая **надежность** устройств и **качество** программирования гарантируются нашим многолетним опытом.

☎ *Звоните нам по телефону* **(095) 246-60-82**

Журнал "Монитор" можно приобрести в городах России, Украины, Белоруссии, Узбекистана, обратившись к нашим официальным дилерам.

1. Самара
МП "Сети", директор Дмитрий Грецов
тел. (8462) 354-932.

2. Волгоград
Арендное предприятие "Техническая книга"
тел. (8442) 363-597.

3. Чебоксары
Фирма "Демос"
тел. (8352) 208-395, 426-492, факс 209-437.

4. Киев
МП "Диас"
тел./факс (044) 220-77-15, тел. 518-80-24.

5. Днепропетровск
Максименко В.И.
тел. (0562) 918-337.

6. Минск
НПП "Триумф"
тел. (0172) 661-609, 661-680, факс 666-606.

7. Ташкент
Центр молодежных исследований
тел. (3712) 688-847.

Листинг

```
{$X+B-}
unit Packer;
interface
  uses Objects;

  const MaxSize       = 4090; {* Число узлов в словаре *}
  const LogOfBound    = 7;    {* 2**7 - число узлов в малом подсловаре *}
  const MaxSuperPack  = 128;
  const MaxPack       = 128;
  const MaxLimit      = 0;

{* Типы функций, с помощью которых читают и записывают исходный
   и упакованный текст. *}

  type Inputter       = function( var C : Char ): Boolean;
       Outputter      = function(     C : Char ): Boolean;

{* Пара процедур, реализующих упаковку-распаковку по алгоритму
   Лемпела-Зива.*}

  procedure FastPack    ( IS : Inputter; OS : Outputter );
  procedure FastUnPack  ( IS : Inputter; OS : Outputter );

{* Пара процедур, реализующих упаковку-распаковку с разрешением
   конфликтов. *}

  procedure SuperPack   ( IS : Inputter; OS : Outputter );
  procedure SuperUnPack ( IS : Inputter; OS : Outputter );

implementation

  const  NullLink = 0;

  type TLink = 0..MaxSize;

  type TLong = record Lo : Char; Hi : Byte; Rest : word end;

  type TChars = array[ 0..MaxSize ] of char;

  type TDir = -1..1;

  const  Left    = -1;
         Down    = 0;
         Right   = 1;

    const InList = $80;

{* Структура узла дерева. Старший бит поля Flags кодирует, включать
   ли узел в список кандидатов на удаление. 6 младших битов кодируют
   номер уровеня узла. *}

  type TNode = record
                 Sym      :  Char;
                 Up       :  TLink;
                 Ref      :  array[ TDir ] of TLink;
                 Prev,Next:  TLink;
                 Count    :  word;
                 Flags    :  byte;
               end;

  type PTree = ^TTree;
       TTree = array[ TLink ] of TNode;

{* Головы и хвосты двунаправленных списков кандидатов на удаление. *}

  type TSubDict = record
                    First:  TLink;
                    Last :  Link;
                  end;

  const Bound = 1 shl LogOfBound;  {* Длина меньшего подсловаря. *}

  type PDict = ^TDict;
       TDict = object( TObject )
                 Count       :  TLink;
                 Small,Large :  TSubDict;
                 Swapped     :  array[ Boolean ] of TLink;
                 Buffer      :  LongInt;
                 BufHead     :  0..31;
                 Tree        :  TTree;
                 constructor Init;
                 function    Add     ( Prefix : TLink; S : Char ) : TLink;
                 function    Delete  ( Prefix : TLink ) : TLink;
                 procedure   Promote ( var Node : TLink );
                 procedure   Correct ( var S : TLink );
                 procedure   PutPosition( OS : Outputter; P : TLink);
                 function    GetPosition( IS : Inputter; var P : TLink ): Boolean;
                 function    PrefixOf ( Node : TLink ): TLink;
                 function    UnCode   ( var Buff; MaxStringSize : word;
                                        Node : TLink ): word;
                 procedure   Exclude  ( Node : TLink );
                 function    Remove   ( R : TLink ): TLink;
                 procedure   Glue     ( R,N : TLink );
                 procedure   Include  ( Node : TLink; UpLink : TLink;
                                        C : Char; Dir : TDir );
                 function    Search   ( Prefix: TLink; S: Char ): TLink;
                 procedure   Renumber( N,R: TLink );
                 procedure   ListAdd  ( var Part : TSubDict; N : TLink );
                 procedure   ListDel  ( var Part : TSubDict; N : TLink );
                 procedure   Enq      ( OS : Outputter; P : word; N : Byte );
                 function    Deq      ( IS : Inputter; var P : word;
                                        N : Byte ): Boolean;
                 procedure   Flq      ( OS : Outputter );
               end;

{*************************************************************
TDict.Init - Инициализация словаря.
Вначале в словаре 256 однобайтовых строк. Они разбрасываются "случайным
образом", чтобы вид дерева был близок к сбалансированому.
*************************************************************}

  constructor TDict.Init;
    const  m = 71;
    var  i : byte;
    var  S : Char;
    var  d : TDir;
    begin      { TDict.Init }
     · TObject.Init;
      S := #217;
      for i := 0 to 255 do begin
        S := Char((Ord(S)+m) mod 256);
        Add( NullLink,S );
      end; { for }
    end;    { TDict.Init }

{*************************************************************
TDict.PrefixOf - Вычисление префикса узла.
Переходим по ссылке UP, пока не перейдем на следующий уровень.
*************************************************************}

  function     TDict.PrefixOf;
    var Temp : TLink;
    begin      { TDict.PrefixOf }
      Temp := Node;
      while Tree[Tree[Temp].Up].Ref[Down] <> Temp do
        Temp := Tree[Temp].Up;
      PrefixOf := Tree[Temp].Up;
    end;    { TDict.PrefixOf }

{*************************************************************
TDict.Glue - Корректировка ссылок Up,Down,Left,Right при удалении
узла с номером R.
*************************************************************}

  procedure TDict.Glue;
    var d : TDir;
    begin { Glue }
      if N <> NullLink then Tree[N].Up := Tree[R].Up;
      with Tree[Tree[R].Up] do begin
        for d := Left to Right do
          if Ref[d] = R then begin
            Ref[d] := N;
            exit
          end; { if }
      end; { with }
    end; { Glue }

{*************************************************************
TDict.Remove - Удаление самого правого узла левого поддерева.
*************************************************************}

  function     TDict.Remove;
    begin   { TDict.Remove }
      while (Tree[R].Ref[Right] <> NullLink) do R := Tree[R].Ref[Right];
      Glue( R,Tree[R].Ref[Left] );
      Remove := R;
    end;    { TDict.Remove }

{*************************************************************
TDict.Exclude - Исключение выбранного узла из дерева.
*************************************************************}

  procedure   TDict.Exclude;
    var Temp : TLink;
    begin   { TDict.Exclude }
      with Tree[Node] do begin
        if Ref[Left ] = NullLink then begin Glue( Node,Ref[Right] ); exit end;
        if Ref[Right] = NullLink then begin Glue( Node,Ref[Left]); exit end;
```

Листинг (продолжение)

```
        Temp := Remove( Ref[Left] );
        Tree[Temp].Ref[Left] := Ref[Left];
        if Ref[Left] <> NullLink then Tree[Ref[Left]].Up := Temp;
        Tree[Temp].Ref[Right] := Ref[Right];
        if Ref[Right] <> NullLink then Tree[Ref[Right]].Up := Temp;
        Glue( Node,Temp );
      end; { with }
    end;    { TDict.Exclude }

{*************************************************************
TDict.Delete - Удаление узла из словаря.
В поисках кандидата на удаление просматривается список наиболее
давно не посещаемых узлов большого (Large) подсловаря.
Выбирается кандидат, удовлетворяющий ограничениям на удаляемый узел.
Узел исключается из дерева.
Если у префикса нет других потомков, он не состоит из одной буквы и
не лежит в малом словаре, то включаем его в список кандидатов на удаление.
Удаляем узел из списка кандидатов на удаление.
*************************************************************}

  function    TDict.Delete;
    var Temp : TLink;
    var UpLink : TLink;
    begin   { TDict.Delete }
      Temp := Large.Last;
      while (Temp <> NullLink) and ((Temp = Prefix) or
        (PrefixOf(Temp) = Prefix)) do
          Temp := Tree[Temp].Prev;
      if Temp <> NullLink then begin
        Exclude( Temp );
        UpLink := Tree[Temp].Up;
        with Tree[UpLink] do
          if ( Ref[Down] = NullLink) and ((Flags and InList) = 0) and
            (( Flags and $7F ) <> 1 ) then begin
              Flags := Flags or InList;
              if UpLink = Bound then ListAdd( Large,UpLink );
          end; { if }
          ListDel( Large,Temp );
      end; { if }
      Delete := Temp;
    end;    { TDict.Delete }

{*************************************************************
TDict.Search - Поиск строки, заданной префиксом и последией буквой.
*************************************************************}

  function    TDict.Search;
    var Temp : TLink;
    begin   { TDict.Search }
      Temp := Tree[Prefix].Ref[Down];
      while (Temp <> NullLink) and (Tree[Temp].Sym <> S ) do
        if Ord(Tree[Temp].Sym) < Ord(S)
          then Temp := Tree[Temp].Ref[Right]
          else Temp := Tree[Temp].Ref[Left];
      Search := Temp;
    end;    { TDict.Search }

{*************************************************************
TDict.Include - Включение нового узла в дерево. Префикс узла при необ-
ходимости исключается из списка на удаление.
*************************************************************}

  procedure   TDict.Include;
    var d : TDir;
    begin   { TDict.Include }
      with Tree[Node] do begin
        for d := Left to Right do Ref[d] := NullLink;
        Sym := C;
        Up  := UpLink;
      end; { with }
      Tree[UpLink].Ref[Dir] := Node;
      Tree[Node].Flags := Tree[Uplink].Flags and $7F;
      if Dir = Down then Inc( Tree[Node].Flags );
      if ((Tree[UpLink].Flags and InList) <> 0) and ( Dir = Down ) then begin
        Tree[UpLink].Flags := Tree[UpLink].Flags and $7F;
        if UpLink > = Bound then ListDel( Large,UpLink );
      end; { if }
      with Tree[Node] do
        if (Flags and $7F) <> 1 then Flags := Flags or InList;
    end;    { TDict.Include }

{*************************************************************
TDict.ListAdd - Включение элемента в список первым.
*************************************************************}

  procedure   TDict.ListAdd( var Part : TSubDict; N : TLink );
    begin   { TDict.ListAdd }
      if Part.First <> NullLink
        then Tree[Part.First].Prev := N
        else Part.Last := N;
      Tree[N].Next := Part.First;
      Tree[N].Prev := NullLink;
      Part.First := N;
    end;    { TDict.ListAdd }

{*************************************************************
Tdict.ListDel - Исключение элемента из списка.
*************************************************************}

  procedure   TDict.ListDel( var Part : TSubDict; N : Tlink );
    begin   { TDict.ListDel }
      if N = Part.Last then Part.Last := Tree[N].Prev;
      if N = Part.First then Part.First := Tree[N].Next;
      if Tree[N].Next <> NullLink then Tree[Tree[N].Next].Prev := Tree[N].Prev;
      if Tree[N].Prev <> NullLink then Tree[Tree[N].Prev].Next := Tree[N].Next;
    end;    { TDict.ListDel }

{*************************************************************
TDict.Add - Создание и включение узла в словарь. Дается префикс и послед-
няя буква. Ранее включенные элементы повторно не включаются. При не-
хватке места в словаре производится исключение.
*************************************************************}

  function    TDict.Add;
    var UpLink  : TLink;
    var Temp    : TLink;
    var d       : TDir;
    begin   { TDict.Add }
      if ((Tree[Prefix].Flags and $7F) > = $3F) then Temp := NullLink
        else begin
          d := Down;
          Temp := Tree[Prefix].Ref[Down];
          UpLink := Prefix;
          while ( Temp <> NullLink ) and ( Tree[Temp].Sym <> S ) do begin
            UpLink := Temp;
            if Ord(Tree[Temp].Sym) < Ord(S)
              then d := 1
              else d := -1;
            Temp := Tree[Temp].Ref[d];
          end; { while }
          if Temp = NullLink then begin
            if (Count = MaxSize) then Temp := Delete( Prefix )
              else begin
                Inc( Count );
                Temp := Count;
            end; { if }
            if Temp <> NullLink then begin
              Include( Temp,UpLink,S,d );
              if Temp < Bound
                then ListAdd( Small,Temp )
                else if (Tree[Temp].Flags and InList) <> 0 then
                  ListAdd( Large,Temp );
              Tree[Temp].Count := 0;
            end; { if }
          end; { if }
      end; { if }
      Add := Temp;
    end;    { TDict.Add }

{*************************************************************
TDict.Promote - Повышение рейтинга узла. Если рейтинг узла из большего
подсловаря превышает рейтинг последнего по списку узла меньшего слова-
ря, то они меняются местами. Номера этих узлов сохраняются в массиве
Swapped для последующей коррекции.
*************************************************************}

  procedure   TDict.Promote;
    var Temp : TLink;
    begin   { TDict.Promote }
      Inc(Tree[Node].Count);
      Swapped[false] := NullLink;
      Swapped[true] := NullLink;
      if Node < Bound then begin
          ListDel( Small,Node );
          ListAdd( Small,Node );
        end
        else begin
          if Tree[Node].Count > Tree[Small.Last].Count then begin
              Temp := Small.Last;
              ListDel( Small,Temp );
              if (Tree[Node].Flags and InList) <> 0 then ListDel( Large,Node );
              Renumber( Temp,Node );
              Swapped[false] := Temp;
              Swapped[true] := Node;
              ListAdd( Small,Temp );
              if (Tree[Node].Flags and InList) <> 0 then ListAdd( Large,Node );
              Node := Temp;
            end
```

```
            else begin
              if (Tree[Node].Flags and InList) <> 0 then begin
                ListDel( Large,Node );
                ListAdd( Large,Node );
              end; { if }
          end { if }
        end; { if }
      end;    { TDict.Promote }

{*************************************************************
TDict.Correct - Коррекция номеров узлов (вне словаря). Применяется после
перестановки.
*************************************************************}

  procedure  TDict.Correct;
    var I : Boolean;
    begin   { TDict.Correct }
      for I := false to true do
        if (Swapped[I] = S) then begin
          S := Swapped[ not I ];
          exit;
        end;
    end;    { TDict.Correct }

{*************************************************************
TDict.Renumber - Перестановка узла из большого подсловаря на место узла из
малого. Корректируются все ссылки внутри словаря.
*************************************************************}

  procedure  TDict.Renumber;
    var Temp : TNode;
    var d    : TDir;
    type TSwap =  record
                    U : TLink;
                    P : array[ TDir ] of TLink;
                    D : TDir
                  end;

    var NEnvir,REnvir : TSwap;

    begin   { TDict.Renumber }
      NEnvir.U := Tree[N].Up;
      REnvir.U := Tree[R].Up;
      for d := Left to Right do begin
        NEnvir.P[d] := Tree[N].Ref[d];
        REnvir.P[d] := Tree[R].Ref[d];
        if Tree[Tree[N].Up].Ref[d] = N then NEnvir.D := d;
        if Tree[Tree[R].Up].Ref[d] = R then REnvir.D := d;
      end; { for }
      if Tree[N].Up = R then Tree[N].Up := N;
      if Tree[R].Up = N then Tree[R].Up := R;
      for d := Left to Right do begin
        if Tree[N].Ref[d] = R then Tree[N].Ref[d] := N;
        if Tree[R].Ref[d] = N then Tree[R].Ref[d] := R;
      end; { for }
      Temp := Tree[N];
      Tree[N] := Tree[R];
      Tree[R] := Temp;
      if NEnvir.U <> R then Tree[NEnvir.U].Ref[NEnvir.D] := R;
      if REnvir.U <> N then Tree[REnvir.U].Ref[REnvir.D] := N;
      for d := Left to Right do begin
        if (NEnvir.P[d] <> NullLink) and (NEnvir.P[d] <> R) then
          Tree[NEnvir.P[d]].Up := R;
        if (REnvir.P[d] <> NullLink) and (REnvir.P[d] <> N) then
          Tree[REnvir.P[d]].Up := N;
      end; { for }
    end;    { TDict.Renumber }

{*************************************************************
TDict.Enq - помещение N битов в выходной буфер и при необходимости сброс
части буфера Buffer в выходной (упакованный) поток.
*************************************************************}

  procedure  TDict.Enq;
    var Temp : LongInt;
    begin   { TDict.Enq }
      Temp := P;
      Temp := Temp shl BufHead;
      Buffer := Temp or Buffer;
      Inc( BufHead,N );
      while (BufHead >= 8) do begin
        OS( TLong( Buffer ).Lo );
        Buffer := Buffer shr 8;
        Dec( BufHead,8 );
      end; { while }
    end;    { TDict.Enq }

{*************************************************************
TDict.Deq - Извлечение N битов из входного потока. Возвращает False, если
входной поток пуст.
*************************************************************}

  function    TDict.Deq;
    const Mask : array[1..14] of LongInt =
      ( $1,$3,$7,$F,$1F,$3F,$7F,$FF,$1FF,$3FF,$7FF,$FFF,$1FFF,$3FFF );
    var Status : Boolean;
    var Temp : LongInt;
    begin   { TDict.Deq }
      Status := true;
      while Status and (BufHead < N) do begin
        Temp := 0;
        Status := IS( TLong(Temp).Lo );
        if Status then begin
          Temp := Temp shl BufHead;
          Buffer := Buffer or Temp;
          Inc( BufHead,8 );
        end { if }
      end; { while }
      P := Buffer and Mask[N];
      Buffer := Buffer shr N;
      Dec( BufHead,N );
      Deq := Status;
    end;    { TDict.Deq }

{*************************************************************
TDict.Flq - Принудительный сброс буфера в выходной поток.
*************************************************************}

  procedure  TDict.Flq;
    begin   { TDict.Flq }
      if BufHead > 0 then OS( TLong( Buffer ).Lo );
    end;    { TDict.Flq }

{*************************************************************
TDict.PutPosition - Вывод позиции. Определяет число битов в кодировке
позиции.
*************************************************************}

  procedure  TDict.PutPosition;
    begin   { TDict.PutPosition }
      if P < 128 then Enq(OS,(P*2)+1,8 ) else Enq( OS,P*2,13 );
    end;    { TDict.PutPosition }

{*************************************************************
Tdict.GetPosition - Ввод позиции. Сначала читается бит длины, а затем номер
позиции. Возвращает False, если входной поток пуст.
*************************************************************}

  function    TDict.GetPosition;
    var Flag : word;
    var Status : Boolean;
    var Cmp : TLink;
    begin   { TDict.GetPosition }
      Status := Deq( IS,Flag,1 );
      if Status then begin
        if (Flag <> 0)
          then Status := Deq( IS,word(P),7 )
          else Status := Deq( IS,word(P),12 );
      end; { if }
      GetPosition := Status;
    end;    { TDict.GetPosition }

{*************************************************************
TDict.UnCode - Проход дерева снизу вверх и помещение в Buff соответствующей
узлу Node строки.
*************************************************************}

  function    TDict.UnCode;
    var  Head   : word;
    var  Temp   : TLink;
    begin   { TDict.UnCode }
      Head := MaxStringSize;
      Temp := Node;
      repeat
        Dec( Head );
        TChars(Buff)[Head] := Tree[Temp].Sym;
        Temp := PrefixOf( Temp );
      until Temp = NullLink;
      UnCode := Head;
    end;    { TDict.UnCode }

{*************************************************************
PutBuffer - Вывод в поток расшифрованной строки из Buff.
*************************************************************}

  procedure  PutBuffer( OS : Outputter;
                        var Buff; Head : word; MaxStringSize : word );
    var I : word;
    begin   { PutBuffer }
      for I := Head to MaxStringSize-1 do OS( TChars(Buff)[Head] );
    end;    { PutBuffer }
```

Листинг (продолжение)

```
{*********************************************************
AppendBuffer - Дописывание первого символа строки в ее конец.
*********************************************************}

  procedure  AppendBuffer( var Buff; var Head : word; MaxStringSize : word );
    var C : char;
    var I : word;
    begin  { AppendBuffer }
      C := TChars(Buff)[Head];
      for I := Head to MaxStringSize-1 do TChars(Buff)[I-1] := TChars(Buff)[I];
      TChars(Buff)[MaxStringSize-1] := C;
      Dec( Head );
    end;   { AppendBuffer }

{*********************************************************
FastPack - Упаковка по методу Лемпела-Зива.
  IS - функция чтения из входного (неупакованного) текста.
  OS - процедура записи упаковки байта.
*********************************************************}

  procedure  FastPack;
    var Dict              : PDict;
    var NotEof            : Boolean;
    var S,B               : Char;
    var Old,Node,Temp : TLink;
    begin  { FastPack }

      {* Инициализация словаря. *}

      Dict := new( PDict,Init );
      if Dict = NIL then exit;
      NotEof := IS( S );

      if NotEof then begin
        Node := Dict^.Search( NullLink,S );
        Dict^.PutPosition( OS, Node );
        Dict^.Promote( Node );
        NotEof := IS( S );

        {* Цикл выделения и кодирования подстрок.*}

        while NotEof do begin
          B := S;
          Old := Node;

          {* Отщепление префикса.*}

          Temp := Dict^.Search( NullLink,S );
          while NotEof and (Temp <> NullLink) do begin
            Node := Temp;
            NotEof := IS( S );
            if NotEof then Temp := Dict^.Search( Node,S );
          end; { while }

          {* Запись позиции словаря в поток и обновление словаря.*}

          if Temp <> NullLink then Dict^.PutPosition( OS,Temp )
            else begin
              Dict^.PutPosition( OS,Node );
              Dict^.Promote( Node );
              if Dict^.Swapped[ false ] <> NullLink then Dict^.Correct( Old );
              Temp := Dict^.Add( Old,B );
            end; { if }
        end; { while }

        {* Очистка буфера битов.*}

        Dict^.Flq( OS );
      end; { if }
      dispose( Dict,Done );
    end;   { FastPack }

{*********************************************************
FastUnPack - Распаковка, работает в паре с FastPack.
  IS - функция чтения байта из входного (упакованного) вида.
  OS - процедура распаковки символа
*********************************************************}

  procedure  FastUnPack;
    var Dict : PDict;
    var NotEof : Boolean;
    var Buff : array[0..MaxPack-1] of char;
    var H : word;
    var A : Char;
    var Old,Node,Temp : TLink;

    begin  { FastUnPack }

      {* Инициализация словаря. *}

      Dict := new( PDict,Init );
      if Dict = NIL then exit;
      Old := NullLink;
      NotEof := Dict^.GetPosition( IS,Node );
      if NotEof then begin
        H := Dict^.UnCode( Buff, MaxPack, Node );
        PutBuffer( OS,Buff,H, MaxPack );

        {* Цикл распаковки. За один виток цикла декодируется одна строка. *}

        repeat
          Dict^.Promote( Node );
          if Dict^.Swapped[false] <> NullLink then Dict^.Correct( Old );
          Temp := Dict^.Add( Old,A );
          Old := Node;
          NotEof := Dict^.GetPosition( IS,Node );
          if NotEof then begin
            H := Dict^.UnCode( Buff, MaxPack, Node );
            PutBuffer( OS,Buff,H, MaxPack );
            A := Buff[H];
          end; { if }
        until not NotEof;

      end; { if }
      dispose( Dict,Done );
    end;   { FastUnPack }

{*********************************************************
SuperPack - Процедура упаковки с разрешением конфликтов.
Параметры те же, что и у FastPack.
*********************************************************}

  procedure  SuperPack;
    type TLine      = 0..MaxSuperPack-1;
    type PConflict  = ^TConflict;
         TConflict  = array[TLine,TLine] of TLink;
    var  Dict       : PDict;
    var  ConfSizes: array[TLine] of integer;
    var  Conf       : PConflict;
    var  TopLine    : TLine;
    var  S          : Char;
    var  NotEof     : Boolean;
    var  ToJump     : Boolean;
    var  NextLines : TLine;
    var  TotalLines: TLine;
    var  Sym        : array[TLine] of Char;
    var  I          : TLine;
    var  Line,Col   : TLine;
    var  Temp       : TLink;
    var  NewLine    : TLine;
    var  MaxSize    : word;
    var  P          : TLink;
    var  OldLine    : TLink;

{*********************************************************
Progress - Выбирает префикса оптимальной длины, кодирование его и сдвиг текущей строки в таблице разрешения конфликтов.
*********************************************************}

  procedure Progress;
    var I,J      : TLine;
    var Local : TLink;
    var Limit  : TLine;
    begin { Progress }

    {* Выбор оптимальной длины префикса. *}

    NewLine := NextLines;
    MaxSize := ConfSizes[ (TopLine+NextLines) mod MaxSuperPack ]+NextLines;
    for I := 1 to NextLines-1 do begin
        Line := (TopLine+I) mod MaxSuperPack;
        if MaxSize < ConfSizes[Line]+I then begin
          MaxSize := ConfSizes[Line]+I;
          NewLine := I;
        end; { if }
    end; { for }

        {* Кодирование выбранного префикса *}

        P := Conf^[ TopLine,NewLine-1 ];
        Dict^.PutPosition( OS,P );

        {* Модификация словаря и корректировка значений в таблице *}
        {* разрешения конфликтов.*}

        Dict^.Promote( P );
        if Dict^.Swapped[false] <> NullLink then begin
          for I := 0 to TotalLines do begin
```

```
          Line := (TopLine+i) mod MaxSuperPack;
          for j := 0 to ConfSizes[Line]-1 do Dict^.Correct( Conf^[Line,j] )
        end; { for }
        if OldLine <> NullLink then Dict^.Correct( OldLine );
      end; { if }
      if OldLine <> NullLink then begin
        Temp := OldLine;

        {* Выбор максимального числа строк, заносимого за один*}
        {* раз в словарь.*}

        if (NewLine-1 < MaxLimit)
          then Limit := NewLine-1
          else Limit := MaxLimit;
        for j := 0 to Limit do begin
          S := Sym[ (TopLine+j) mod MaxSuperPack ];
          Local := Dict^.Search( Temp,S );
          if Local <> NullLink then Temp := Local
            else begin
            Temp := Dict^.Add( Temp,S );
            if Temp <> NullLink then begin
              for i := 1 to TotalLines do begin
                Line := (TopLine+i) mod MaxSuperPack;
                if Temp = Conf^[Line,ConfSizes[Line]-1] then begin
                  Conf^[Line,ConfSizes[Line]-1] := NullLink;
                  Dec( ConfSizes[Line] );
                end; { if }
              end; { for }
            end; { if }
          end; { if }
        end; { for }
      end; { if }

      OldLine := Conf^[TopLine,NewLine-1];

      {* Продвижение текущей строки (TopLine) *}

      TopLine := (TopLine+NewLine) mod MaxSuperPack;
      NextLines := ConfSizes[ TopLine ];
      Dec(TotalLines,NewLine);
    end; { Progress }

  begin   { SuperPack }

    {* Инициализация словаря *}

    Dict := new( PDict,Init );
    if Dict = NIL then exit;

    {* Инициализация таблицы разрешения конфликтов *}

    new( Conf );
    if Conf = NIL then begin
      dispose( Dict );
      exit;
    end; { if }
    TopLine := 0;
    FillChar( ConfSizes,SizeOf( ConfSizes ),#0 );
    FillChar( Sym,SizeOf( Sym ),#0 );
    NotEof := IS( Sym[TopLine] );
    if NotEof then begin
      S := Sym[ TopLine ];
      Conf^[TopLine,0] := Dict^.Search( NullLink,S );
      Inc( ConfSizes[ TopLine ] );
      NextLines := 1;
      TotalLines := 1;
      OldLine := NullLink;
      NotEof := IS( Sym[ TotalLines ] );

      {* Цикл по всем символам входного текста. *}

      while NotEof do begin
        ToJump := true;

        {* Прибавление вновь считанного символа ко всем префиксам *}

        for i := 1 to TotalLines do begin
          Line := (TopLine+i) mod MaxSuperPack;
          Col := (TotalLines-i);
          S := Sym[ (TopLine+TotalLines) mod MaxSuperPack ];
          if (Col = 0) then begin
              ConfSizes[ Line ] := 0;
              Temp := Dict^.Search( NullLink,S);
            end
            else if Conf^[Line,Col-1] = NullLink
              then Temp := NullLink
              else Temp := Dict^.Search( Conf^[Line,Col-1],S);
          if (i <= NextLines ) then ToJump := ToJump and (Temp = NullLink);
          Conf^[Line,Col] := Temp;
          if Temp <> NullLink then Inc( ConfSizes[ Line ]);
        end; { for }

        {* Если длины строк, идущих за префиксом, определились,*}
        {* то можно выбрать и закодировать префикс оптимальной*}
        {* длины, продвинув текущую позицию в таблице разрешения*}
        {* конфликтов.*}

        if ToJump then Progress;

        {* На каждом шаге число возможных префиксов увеличивается*}
        {* на единицу.*}

        Inc( TotalLines );
        NotEof := IS ( Sym[(TopLine+TotalLines) mod MaxSuperPack] );
      end; { while }

      {* Выталкивание последнего префикса. *}

      Progress;
      if ConfSizes[ TopLine ] <> 0 then begin
        Dict^.PutPosition( OS,Conf^[TopLine,ConfSizes[ TopLine ]-1] );
      end; { if }

      {*.Очистка буфера битов *}

      Dict^.Flq( OS );
    end; { if }
    dispose( Conf );
    dispose( Dict,Done );
  end;    { SuperPack }

{*********************************************************
SuperUnPack - Процедура распаковки. Работает в паре с SuperPack.
Параметры имеют тот же смысл, что и у FastUnPack.
*********************************************************}

  procedure   SuperUnPack;
    var Dict     : PDict;
    var NotEof   : Boolean;
    var P        : TLink;
    var Node     : TLink;
    var Temp     : TLink;
    var i        : byte;
    var B        : Char;
    var Buff     : array[0..MaxSuperPack-1] of char;
    var H        : word;
    var Limit    : word;

    begin   { SuperUnPack }

      {* Инициализация словаря *}

      Dict := new( PDict,Init );
      if Dict = NIL then exit;
      NotEof := Dict^.GetPosition( IS,P );
      if NotEof then begin
        H := Dict^.UnCode( Buff,MaxSuperPack,P );
        PutBuffer( OS, Buff, H, MaxSuperPack );
        Dict^.Promote( P );
        Node := P;
        NotEof := Dict^.GetPosition( IS,P );

        {* Цикл распаковки. За один виток цикла декодируется одна строка. *}

        while NotEof do begin
          H := Dict^.UnCode( Buff,MaxSuperPack,P );
          PutBuffer( OS, Buff, H, MaxSuperPack );
          Dict^.Promote( P );
          if Dict^.Swapped[false] <> NullLink then Dict^.Correct( Node );

          {* Выбор максимального числа строк, заносимого за один*}
          {* раз в словарь.*}

          if (MaxSuperPack-1-H < MaxLimit)
            then Limit := MaxSuperPack-1
            else Limit := H+MaxLimit;
          for i := H to Limit do begin
            B := Buff[i];
            Node := Dict^.Add( Node,B );
          end; { for }
          Node := P;
          NotEof := Dict^.GetPosition( IS,P );
        end; { while }

      end; { if }
      dispose( Dict,Done );
    end;    { SuperUnPack }

end. { Packer }
```

Трехмерная графика

Д. Солдатенков

– Куб ... ну это как бы выпуклый квадрат ...

Из объяснений лектора
по курсу стереометрии

Термин "компьютерная графика" относится к особому типу изображений, создаваемых на компьютере. На самом деле эта графика, конечно, двумерная, то есть создаваемая на плоском экране или листе бумаги, но таким образом, что воображение без труда восстанавливает пространственную структуру изображаемого объекта. Подобного рода изображения создавались художниками и фотографами с незапамятных времен, а компьютер позволил лишь автоматизировать и ускорить этот процесс. Правда, до появления компьютеров никто не называл эти картинки трехмерными.

Свое название "трехмерные" компьютерные изображения получили потому, что создаются они при помощи проектирования трехмерных объектов, хранящихся в памяти, на плоскость экрана или листа бумаги.

Трехмерная графика используется почти во всех классах программного обеспечения, начиная с систем автоматического проектирования и кончая компьютерными играми. Игра "F-19", являясь яркой представительницей трехмерных игр, до сих пор остается одной из самых популярных.

Различные задачи предъявляют неодинаковые требования к изображениям, а разнообразие требований порождает несхожесть алгоритмов. Наиболее древние проблемы в области создания трехмерных изображений связаны с алгоритмами уничтожения невидимых линий и окрашивания.

В задачах автоматизированного проектирования изображения строятся таким образом, чтобы удобно было их выводить на графопостроитель, поэтому вместо растрового заполнения используются различные штриховки. И алгоритмы уничтожения невидимых линий не используют дискретности изображения или перерисовки одних элементов поверх других.

В последнее время стал особенно популярен метод обратного хода луча, при использовании которого прослеживается путь гипотетического луча света, который выходит из глаза наблюдателя, проходит через пиксел экрана и попадает в те трехмерные объекты, присутствие которых предполагается за экраном. При этом луч преломляется и переотражается, окрашивая надлежащим образом тот пиксел, через который он прошел в начале. Самой популярной программой, позволяющей создавать изображения этим методом, является 3D-Studio.

Метод обратного хода луча связан с огромными объемами вычислений, поэтому в реальном времени его пока использовать нельзя, но изображение, получающееся при этом, невозможно отличить от фотографии. Такие изображения используются в видеоклипах и для изготовления заставок и коротких мультфильмов.

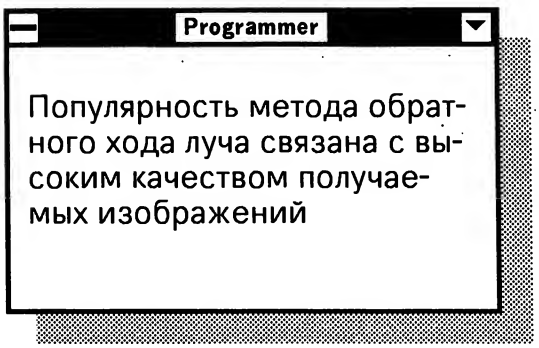
Компьютерные игры используют самый быстрый способ создания трехмерных изображений. Основная идея этого способа заключается в том, что с некоторой потерей качества любой объект можно представить как совокупность многоугольников. Далее предполагается, что при центральном проектировании на плоскость экрана прямые линии переходят в прямые. В этом случае можно рассчитать только координаты проекций вершин многоугольника, затем соединить их отрезками так, как они соединены в пространстве, и мы получим изображение многоугольника, которое будет тоже многоугольником.

Что касается алгоритма удаления невидимых линий, то он тоже достаточно прост. Если выводить многоугольники в порядке их приближения к наблюдателю, то те, которые должны быть видимыми, сами закроют те, которые должны быть невидимыми. Правда, если многоугольники недостаточно малы, могут возникнуть ошибки.

Освещенность граней объекта можно принимать во внимание, рассчитывая угол между нормалью к грани и направлением падения света на эту грань.

Приведенная в листинге 1 программа строит изображение объекта, которым в данном случае является поверхность, состоящая из треугольников. Задается положение источника света и рассчитывается освещенность граней. При выводе на экран используется только черный и белый цвет: такое изображение выводится на печать без искажений. Программу удобно использовать для изображения функций двух переменных. Можно менять точку зрения и положение источника света.

Рассмотрим подробнее работу этой программы. Она написана на языке Турбо Паскаль и использует только его стандартные графические средства. Графический



режим - EGA 640x350, но при желании может быть использован любой другой.

Координатные оси в пространстве выбираются следующим образом: X - снизу вверх, Y - слева направо, Z - вперед. Координатные оси на экране располагаются как обычно.

В массиве IniPoi^ хранятся начальные пространственные координаты точек. Этот массив формируется процедурой FormIniPoi, в которой Y и Z распределяются по координатной сетке как независимые переменные, а X является их функцией.

Процедура FormPoi может поворачивать начальный массив точек на произвольный угол и смещать его в пространстве. Это достигается простым домножением координат на матрицу из синусов и косинусов и прибавлением константы, правда, в данной программе для простоты реализовано вращение только вокруг оси X. Преобразованные координаты точек содержатся в массиве Poi^.

В процедуре FormPl на каждые три точки, лежащие рядом, натягивается плоскость и так образуется поверхность, составленная из маленьких треугольничков.

Будем считать, что плоскость экрана проходит через начало координат и перпендикулярна оси Z, тогда уравнение этой плоскости Z=0. Для каждой точки изображаемого объекта рассчитываем уравнение прямой, проходящей через эту точку и через точку, в которой находится наблюдатель. Эта прямая и будет являться лучом зрения. Чтобы получить координаты точки пересечения плоскости экрана с лучом зрения, надо подставить в систему уравнений этой прямой Z=0. Таким образом мы получим изображения точек объекта. Система уравнений, описывающая прямую, проходящую через две точки, выглядит так:

$$\frac{x-X1}{X1-X2}=\frac{y-Y1}{Y1-Y2}=\frac{z-Z1}{Z1-Z2}$$

где X1, Y1, Z1 - координаты первой точки, X2, Y2, Z2 - второй.

Для того чтобы получить изображение треугольника, находящегося в пространстве, надо соединить прямыми изображения его вершин на плоскости точно так же, как они соединены прямыми в пространстве. Результат этой операции представлен на рис. 1.

Если хочется удалить невидимые линии, то треугольники надо рисовать заполненными и выводить их в порядке уменьшения расстояния от них до наблюдателя. Для этого сначала вызывается процедура CalcDist, которая рассчитывает расстояния от треугольников до наблюдателя, а затем процедура SortPlane сортирует треугольники по близости к наблюдателю. Для сортировки треугольников заведен специальный сцепленный список.

Когда новый элемент вставляется в уже отсортированную часть списка, то никаких передвижений в списке не происходит. Программа просто перенаправляет указатели двух соседних элементов на вставляемый элемент, а указатели вставляемого элемента - на элементы, которые были соседними.

Процедура DrawPlan перебирает элементы списка и выводит треугольнички в порядке их приближения. Результат этой операции представлен на рис. 2.

Теперь можно перейти к освещенности объекта. Если предположить, что освещенность плоскости зависит от угла падения света на эту плоскость, то можно достаточно просто рассчитать цвет каждого треугольничка.

Сначала найдем уравнение плоскости Ax + By + + Cz + D = 0, включающей в себя треугольник. Это можно сделать, воспользовавшись двумя условиями:

$$\begin{vmatrix} x & y & z \\ (X2-X1) & (Y2-Y1) & (Z2-Z1) \\ (X3-X1) & (Y3-Y1) & (Z3-Z1) \end{vmatrix} = Ax+By+Cz,$$

$$A*X1+B*Y1+C*Z1+D=0,$$

где X1, Y1, Z1 - координаты первой точки, и так далее. Вычисление коэффициентов для уравнения плоскости происходит в процедуре GetEqu. Затем вычислим вектор нормали к плоскости. Этим занимается процедура GetNormVect. Потом процедура GetLiteVect находит вектор света, который направлен от источника на треугольник. Вычислив скалярное произведение этих векторов и разделив его на длины умножаемых векторов, мы получим косинус угла между ними. Это происходит в процедуре GetCosA.

Теперь остается неясным, с какой стороны падает свет на плоскость. Если с видимой, то чем меньше угол падения, тем плоскость светлее, а если с невидимой, то темнее. Определить направление падения света можно, подставив координаты наблюдателя и источника света в уравнение плоскости. Если получившиеся значения имеют одинаковые знаки, то точки лежат в одном полупространстве и свет падает с видимой стороны, если же нет, то с невидимой. Выяснением этого обстоятельства занимается функция GetSign.

Функция GetColor окончательно рассчитывает цвета треугольничков. Процедура вывода ставит в соответствие цвету структуру из массива Pattern (листинг 2), который описан как процедура, для того чтобы можно было описывать отдельные биты на языке ассемблера. Эта процедура выделена в отдельный файл.

Результат работы программы с расчетом освещенности плоскостей показан на рис. 3. А если отключить изображение ребер треугольничков, то получится довольно гладкая поверхность (рис.4), которая и изображается программой, приведенной в листинге 1. ■

Рис. 1

Рис. 2

Рис. 3

Рис. 4

Листинг 1

```
uses Graph, Crt;

const
     Nz = 33;                              { Число точек по Z }
     Ny = 33;                              { Число точек по Y }
     NOfPoint = Ny * Nz;                   { Общее число точек }
     NOfPlane = (Ny-1) * (Nz-1) * 2;       { Число плоскостей }

type
     TLoc = record                         { Точка в пространстве }
          X, Y, Z: Real;
     end;
     PLocArr = ^TLocArr;
     TLocArr = array [1..NOfPoint] of TLoc; { Массив точек в пространстве }
     PScArr = ^TScArr;
     TScArr = array [1..NOfPoint] of PointType; { Массив точек на экране }
     TPlane = record                       {Плоскость}
          P: array [1..3] of Word;         { натянута на 3 точки }
          Color: Byte;                     { Цвет плоскости}
          Dist: Real;                      { Расстояние до плоскости}
     end;
     PPlaneArr = ^TPlaneArr;
     TPlaneArr = array [1..NOfPlane] of TPlane; { Массив плоскостей }
     PRang = ^TRang;
     TRang = record      { Тип для сортировки плоскостей по удаленности }
          Prev, Next: PRang;
          Val: Word;
     end;
     TEqu = record            { Уравнение плоскости в пространстве }
          A, B, C, D: Real;
     end;
     PPattArr = ^TPattArr;
     TPattArr = array [0..16] of FillPatternType; { Окраска плоскостей }

const
     GrToRad: Real = 3.1416/180;           { Перевод градусов в радианы }
     GD: Integer = 3;                      { Графический режим }
     GM: Integer = 1;
     VPage0: Pointer = Ptr ($A000, 0);{ Указатели на видеостраницы }
     VPage1: Pointer = Ptr ($A800, 0);
     Look: TLoc = (X : 0; Y : 0; Z :-350);      { Положение наблюдателя }
     Lite: TLoc = (X : 50; Y : 200; Z :-20); { Положение источника света }

var
     Rang, OldRang: array [1..NOfPlane] of TRang; { Массивы для сортировки
                          плоскостей по расстоянию до наблюдателя }
     First, Last: TRang;       { Первый и последний элементы в списке плоскостей }
     IniPol: PLocArr;          { Указатель на массив начальных точек }
     Plane: PPlaneArr;         { Указатель на массив плоскостей }
     Pol: PLocArr;             { Указатель на массив развернутых точек }
     ScPol: PScArr;            { Указатель на массив точек на экране }
     Angle: Integer;           { Угол поворота поверхности }
     Ch: Char;

{$I PATTERNS}          { Массив струкрур для раскраски плоскостей }

procedure FormPl; { Натягиваем плоскости на точки }
var
     i, j, k, n: Integer;
begin
     k := 1; n := 1;
     for j:= 1 to Nz-1 do
     for i:= 1 to Ny-1 do begin
          with Plane^[k] do begin
               P[1] := n + (n-1) div (Ny-1) + Ny;
               P[2] := n + (n-1) div (Ny-1);
               P[3] := n + (n-1) div (Ny-1) + 1;
          end;
          Inc (k);
          with Plane^[k] do begin
               P[1] := n + (n-1) div (Ny-1) + 1;
               P[2] := n + (n-1) div (Ny-1) + Ny;
               P[3] := n + (n-1) div (Ny-1) + Ny + 1;
          end;
          Inc (k); Inc (n);
     end;
end;

procedure FormIniPol; { Формируем массив начальных точек }
var
     i, j, k: Integer;
     ForY, ForZ: Real;
begin
     k := 1;
     for j:= 1 to Nz do
          for i:= 1 to Ny do begin
               ForY := i - Ny div 2 - 1; { Смещаем центр начала координат }
               ForZ := j - Nz div 2 - 1;
               with IniPol^[k] do begin
                    X := 25 * Sin (Sqrt (ForY*ForY + ForZ*ForZ) / 2.7);
                                   { X - функция от Y и Z }
                    Y := ForY * 250 / Ny; { Y и Z - просто координатная сетка }
                    Z := ForZ * 250 / Nz;
               end;
               Inc (k);
          end;
end;

function PlanX ( X, Z: Real): Integer;
{ Рассчитываем X пересечения луча зрения и плоскости экрана }
begin
     PlanX := Round (Look.Z * X / (Look.Z - Z));
end;

function PlanY ( Y, Z: Real): Integer;
{ Рассчитываем Y пересечения луча зрения и плоскости экрана }
begin
     PlanY := Round (Look.Z * Y / (Look.Z - Z));
end;

procedure OnScreen ( X, Y: Integer; var ScrX, ScrY: Integer);
{ Рассчитываем координаты точки на экране }
begin
     ScrX := Y + 320;
     ScrY := 75 - Round (0.8 * X);
end;

procedure FormPol (Angle, dX, dY, dZ: Integer);
{ Поворот и смещение начального массива точек }
var
     i: Integer;
     RealA, CosA, SinA: Real;
begin
     RealA := Angle * GrToRad;
     CosA := Cos (RealA);
     SinA := Sin (RealA);
     for i:= 1 to NOfPoint do
          with IniPol^[i] do begin
               Pol^[i].X := X + dX;
               Pol^[i].Z := Z * CosA - Y * SinA + dZ;
               Pol^[i].Y := Z * SinA + Y * CosA + dY;
          end;
end;

procedure FormScPol;
{ Формируем массив точек на экране }
var
     i: Integer;
begin
     for i:= 1 to NOfPoint do
          with Pol^[i] do
               OnScreen ( PlanX (X, Z), PlanY (Y ,Z), ScPol^[i].X, ScPol^[i].Y);
end;

procedure CalcDist;
{ Рассчитываем расстояние от плоскости до наблюдателя }
var
     i, j: Integer;
     tX, tY, tZ: Real;
begin
     for i:= 1 to NOfPlane do begin
          tX := 0; tY := 0; tZ := 0;
          for j:= 1 to 3 do
               with Pol^[Plane^[i].P[j]] do begin
                    tX := tX + X; tY := tY + Y; tZ := tZ + Z;
               end;
          tX := tX / 3; tY := tY / 3; tZ := tZ / 3;
          tZ := tZ - Look.Z;
          Plane^[i].Dist := tX*tX + tY*tY + tZ*tZ;
     end;
end;

procedure InitSort;
{ Инициализация массива для сортировки плоскостей }
var i: Word;
begin
     First.Prev := NIL;
     First.Next := @Rang[1];
     Rang[1].Prev := @First;
     for i:= 1 to NOfPlane-1 do begin
          Rang [i].Val := i;
          Rang [i].Next := @Rang [i+1];
```

```
            Rang [i+1].Prev := @Rang [i];
        end;
        Last.Next := NIL;
        Last.Prev := @Rang[NOfPlane];
        Rang [NOfPlane].Val := NOfPlane;
        Rang [NOfPlane].Next := @Last;
end;

procedure SortPlan;
{ Сортировка плоскостей по расстоянию до наблюдателя }
var
        i, j, k: Word;
        Val: Word;
        Test: PRang;
begin
        Move (Rang, OldRang, NOfPlane*SizeOf (TRang));
        Val := Last.Prev^.Val;
        First.Next := @Rang[Val];
        Rang[Val].Prev := @First;
        while OldRang[Val].Prev^.Prev  NIL do begin
                Val := OldRang[Val].Prev^.Val;
                Test := First.Next;
                while (Test^.Next  NIL) and
                (Plane^[Val].Dist  Plane^[Test^.Val].Dist) do Test := Test^.Next;
                Test^.Prev^.Next := @Rang [Val];
                Rang[Val].Prev := Test^.Prev;
                Rang[Val].Next := Test;
                Test^.Prev := @Rang [Val];
        end;
end;

procedure DrawPlan;
{ Рисуем плоскости в порядке приближения к наблюдателю }
var
        i, j: Integer;
        P: array [1..3] of PointType;
        Val: Word;
begin
        Val := Last.Prev^.Val;
        repeat
                for j := 1 to 3 do P[j] := ScPol^[Plane^[Val].P[j]];
                SetFillPattern (PPattArr (@Pattern)^[Plane^[Val].Color], 15);
                FillPoly (3, P);
                if Rang[Val].Prev^.Prev = NIL then Exit;
                Val := Rang[Val].Prev^.Val;
        until FALSE;
end;

procedure Show;
{ Показываем невидимую страницу }
begin
        Port [$3CE] := 5;
        Port [$3CF] := 1;
        Move (VPage1^, VPage0^, 80*350);
        Port [$3CE] := 5;
        Port [$3CF] := 0;
end;

procedure GetNormVect (Equ: TEqu; var Loc: TLoc);
{ Определяем нормаль к плоскости }
begin
        with Loc, Equ do begin
                X := A; Y := B; Z := C;
        end;
end;

procedure GetLiteVect (Plane: TPlane; var Loc: TLoc);
{ Определяем вектор света }
begin
        with Loc do begin
                X := Pol^[Plane.P[1]].X - Lite.X;
                Y := Pol^[Plane.P[1]].Y - Lite.Y;
                Z := Pol^[Plane.P[1]].Z - Lite.Z;
        end;
end;

function GetCosA (a, b: TLoc): Real;
{ Определяем угол падения света на плоскость }
var
        Moda, Modb: Real;
begin
        with a do Moda := Sqrt (X*X+Y*Y+Z*Z);
        with b do Modb := Sqrt (X*X+Y*Y+Z*Z);
        GetCosA := (a.X*b.X+a.Y*b.Y+a.Z*b.Z)/(Moda*Modb);
end;

function GetColor (CosA: Real; Sign: Integer): Byte;
{ Определяем цвет плоскости }
begin
        if CosA  0 then CosA := -CosA;
        if Sign  0 then GetColor := Round (8 - CosA * 8)
        else GetColor := Round (8 + CosA * 8);
end;

procedure GetEqu (var Equ: TEqu; Plane: TPlane);
{ Определяем уравнение плоскости }
var
        e: array [1..6] of Real;
begin
        with Equ, Pol^[Plane.P[1]] do begin
                e[1] := Pol^[Plane.P[2]].X - X;
                e[2] := Pol^[Plane.P[2]].Y - Y;
                e[3] := Pol^[Plane.P[2]].Z - Z;
                e[4] := Pol^[Plane.P[3]].X - X;
                e[5] := Pol^[Plane.P[3]].Y - Y;
                e[6] := Pol^[Plane.P[3]].Z - Z;
                A := e[2]*e[6] - e[3]*e[5];
                B := e[3]*e[4] - e[1]*e[6];
                C := e[1]*e[5] - e[2]*e[4];
                D := - A*X - B*Y - C*Z;
        end;
end;

function GetSign (var Equ: TEqu; Loc: TLoc): Integer;
{ Определяем полупространство }
var
        n: Real;
begin
        with Loc, Equ do n := A*X + B*Y + C*Z + D;
        if n  0 then
                GetSign := 1
        else
                GetSign := -1;
end;

procedure SetColors;
{ Расставляем цвета плоскостей }
var
        i: Integer;
        Equ: TEqu;
        NormVect, LiteVect: TLoc;
        Sign: Integer;
begin
        for i := 1 to NOfPlane do begin
                GetEqu (Equ, Plane^[i]);
                GetNormVect (Equ, NormVect);
                GetLiteVect (Plane^[i], LiteVect);
                Sign := GetSign (Equ, Lite) - GetSign (Equ, Look);
                Plane^[i].Color :=
                GetColor (GetCosA (NormVect, LiteVect), Sign);
        end;
end;

procedure TakeMem;
{ Резервируем память }
begin
        GetMem (IniPol, SizeOf (TLocArr));
        GetMem (Plane, SizeOf (TPlaneArr));
        GetMem (Pol, SizeOf (TLocArr));
        GetMem (ScPol, SizeOf (TScArr));
end;

procedure PutMem;
{ Освобождаем память }
begin
        FreeMem (IniPol, SizeOf (TLocArr));
        FreeMem (Plane, SizeOf (TPlaneArr));
        FreeMem (Pol, SizeOf (TLocArr));
        FreeMem (ScPol, SizeOf (TScArr));
end;

begin
        InitGraph (GD, GM, '');
        SetActivePage (1);
        SetColor (0);
        TakeMem;
        FormIniPol;
        FormPl;
        InitSort;
        Angle := 0;
        repeat
                SetFillStyle (1, 15);
                Bar (0, 0, 639, 349);
                FormPol (Angle, -70, 0, -60);
```

Листинг 1 (продолжение)

```
            FormScPoi;
            CalcDist;
            SortPlan;
            SetColors;
            DrawPlan;
            Show;
            Angle:= Angle + 5; if Angle  360 then Angle := Angle - 360;
        until KeyPressed;
        Ch := ReadKey;
        PutMem;
        CloseGraph;
    end.
```

Листинг 2

```
    procedure Pattern; assembler;
    asm
        {0}
        db  11111111b
        db  11111111b
        db  11111111b
        db  11111111b
        db  11111111b
        db  11111111b
        db  11111111b
        db  11111111b
        {1}
        db  11111111b
        db  11111111b
        db  11111111b
        db  11101110b
        db  11111111b
        db  11111111b
        db  11111111b
        db  11101110b
        {2}
        db  11111111b
        db  10111011b
        db  11111111b
        db  11101110b
        db  11111111b
        db  10111011b
        db  11111111b
        db  11101110b
        {3}
        db  11111111b
        db  10101010b
        db  11111111b
        db  11101110b
        db  11111111b
        db  10101010b
        db  11111111b
        db  11101110b
        {4}
        db  11111111b
        db  10101010b
        db  11111111b
        db  10101010b
        db  11111111b
        db  10101010b
        db  11111111b
        db  10101010b
        {5}
        db  11111111b
        db  10101010b
        db  11011101b
        db  10101010b
        db  11111111b
        db  10101010b
        db  11011101b
        db  10101010b
        {6}
        db  11011101b
        db  10101010b
        db  11011101b
        db  10101010b
        db  11011101b
        db  10101010b
        db  11011101b
        db  10101010b
        {7}
        db  01010101b
        db  10101010b
        db  11011101b
        db  10101010b
        db  01010101b
        db  10101010b
        db  11011101b
        db  10101010b
        {8}
        db  01010101b
        db  10101010b
        db  01010101b
        db  10101010b
        db  01010101b
        db  10101010b
        db  01010101b
        db  10101010b
        {9}
        db  10101010b
        db  01010101b
        db  00100010b
        db  01010101b
        db  10101010b
        db  01010101b
        db  00100010b
        db  01010101b
        {10}
        db  00100010b
        db  01010101b
        db  00100010b
        db  01010101b
        db  00100010b
        db  01010101b
        db  00100010b
        db  01010101b
        {11}
        db  00000000b
        db  01010101b
        db  00100010b
        db  01010101b
        db  00000000b
        db  01010101b
        db  00100010b
        db  01010101b
        {12}
        db  00000000b
        db  01010101b
        db  00000000b
        db  01010101b
        db  00000000b
        db  01010101b
        db  00000000b
        db  01010101b
        {13}
        db  00000000b
        db  01010101b
        db  00000000b
        db  00010001b
        db  00000000b
        db  01010101b
        db  00000000b
        db  00010001b
        {14}
        db  00000000b
        db  01000100b
        db  00000000b
        db  00010001b
        db  00000000b
        db  01000100b
        db  00000000b
        db  00010001b
        {15}
        db  00000000b
        db  00000000b
        db  00000000b
        db  00010001b
        db  00000000b
        db  00000000b
        db  00000000b
        db  00010001b
        {16}
        db  00000000b
        db  00000000b
        db  00000000b
        db  00000000b
        db  00000000b
        db  00000000b
        db  00000000b
        db  00000000b
    end;
```

Модули расширения языка Clarion

Л. Ливеровский

Прежде чем приступить к рассмотрению структуры модулей расширения языка Clarion, или иначе LEM модулей (LEM — language extension module), определим назначение LEM модулей и покажем необходимость включения в инструментальную среду программирования (ИСП) Clarion этого элемента.

В системе Clarion модули расширения языка выполняют ту же функцию, что и процедурные библиотеки в других языках программирования. Существенное отличие от других языков программирования состоит в том, что библиотеки процедур в других языках программирования могут быть написаны как на самом этом языке, так и на других языках программирования, а в ИСП Clarion LEM модуль не может быть составлен на языке Clarion.

Использование LEM модулей, имеющих особую структуру, отличающуюся от обычных библиотек, диктуется следующим.

Программы, созданные в ИСП Clarion, могут выполняться в режиме интерпретации. Интерпретируется программой Processor некий псевдокод, созданный программой-компилятором Compiler из исходного текста. При этом, естественно, к программе нельзя стандартной процедурой компоновки подключить модули из обычных библиотек традиционных языков программирования, таких как C.

Получение выполняемой программы в EXE формате в ИСП Clarion состоит из трех шагов: компиляция исходного текста в интерпретируемый псевдокод, его трансляция и компоновка. При этом необходимо обеспечить пользователя механизмом создания библиотек подпрограмм, одинаково работающих и под управлением Processor и в составе EXE программы. Наличие двух сред с разной организацией, в которых одинаково должны работать подпрограммы из LEM модуля, по мнению автора, является основной причиной ограничений, накладываемых на LEM модуль. Наиболее сильные ограничения на написание набора подпрограмм накладывает необходимость обеспечить их доступность для вызова из интерпретатора.

Разрешение внешних ссылок (ссылок на глобальные данные) и настройка адресов в ИСП Clarion производятся на этапе компиляции по собственным внутренним правилам Clarion, понятным интерпретатору Processor. Поэтому подключаемый внешний модуль должен работать без настройки адресов в том месте памяти, где он окажется или куда его загрузит интерпретатор Processor. Этим определяется следующее требование: подпрограммы, помещаемые в LEM модуль, должны допускать преобразование в двоичный позиционно независимый формат образа памяти. Требование возможности преобразования LEM модуля в позиционно независимый формат налагает на него следующие основные ограничения:

- размер результирующего файла образа памяти, содержащего код и данные, определенные на этапе компиляции, должен быть менее 64Кбайтов. Это не означает, однако, что подпрограммы, входящие в состав LEM модуля, не могут обращаться к памяти за пределами своих 64Кбайтов;
- нельзя использовать сегмент типа STACK. То есть выделение памяти под стек и загрузка регистров процессора SS:SP соответствующими значениями должна осуществляться программистом или возможно использование стека Clarion;
- в программе на языке ассемблера нельзя использовать символические имена, представляющие адреса сегментов (сегментную часть адреса) в качестве операндов команд, требующих непосредственного значения (как явно, так и косвенно). Если использовать такие конструкции, то для правильной работы такая программа потребует не только

загрузки сегментных регистров необходимыми значениями, но и корректировки значений сегментных адресов в ячейках памяти после загрузки программы в память (segment fixup). Такая программа не будет позиционно независимой.

Недопустимые конструкции приведены в примере 1.

LEM модуль нельзя помещать в сегменты перекрытий (overlay segments).

Из LEM модуля нельзя ссылаться ни на какие имена, определенные в программе на Clarion (обращаться к глобальным переменным и вызывать его функции).

Во всяком случае, в документации по Clarion механизмы доступа из LEM модулей к глобальным переменным и процедурам в программе на Clarion не описаны. Все взаимодействие подпрограмм из LEM модуля и основной программы осуществляется через механизм передачи параметров, описанный ниже.

Однако нет оснований полагать, что нельзя создать такие объектные модули, прикомпоновываемые к EXE программе на Clarion, которые дадут доступ из них ко всей среде Clarion. Ведь весь Clarion и его библиотеки написаны на Turbo C и TASM.

И действительно, существуют разработки, дающие доступ к среде Clarion таким способом. (Но используя такие механизмы, нельзя работать с Processor, в состав которого интегрирован отладчик. Увы, если где-то прибыло, то где-то и убыло.)

Структура LEM модуля

LEM модуль состоит из заголовка, одного или более блоков описания подпрограмм, которые могут содержать в себе необходимое количество блоков описания параметров подпрограмм, текста подпрограмм и признака конца модуля.

Структура LEM модуля приведена на рис. 1, структура заголовка - в таблице 1, блока описания подпрограммы - в таблице 2, блока описания параметра - в таблице 3, признака конца LEM - в таблице 4.

Пример 1. Некоторые недопустимые конструкции

```
SegName   SEGMENT    PARA  PUBLIC
          ASSUME     CS:SegName
Data      DB         'Данные',0
Prc       PROC       FAR
          .
L:
          .
          RET
Prc       ENDP
SegName   ENDS
CodSeg    SEGMENT    PARA  PUBLIC   'CODE'
          .
          MOV        AX,SegName  ;Явная ссылка на,
          .                      ;сегментный адрес
          MOV        AX,SEG Data ;Косвенная ссылка
          .                      ;на сегментный адрес
          CALL       Prc         ;Косвенная ссылка на
          .                      ;сегментный адрес
          JMP        FAR PTR L   ;Косвенная ссылка на
          .                      ;сегментный адрес
CodSeg    ENDS
```

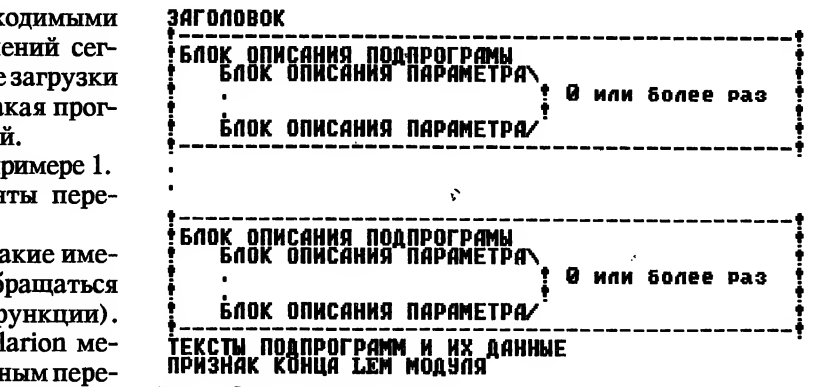


Рис. 1. Структура LEM модуля

Вызов подпрограмм из LEM модуля осуществляется исполняющей системой Clarion передачей управления в точку, определяемую содержимым слова "смещение точки входа" соответствующего блока описания под-

Таблица 1

Заголовок

Смещение	Длина	Тип	
0	3	Байт	'BIO' – Сигнатура
3	4		Резерв
7	2	Слово	Длина LEM модуля
9	1	Байт	Количество подпрограмм в LEM модуле

Таблица 2

Блок описания программы (Routine Descriptor)

Смещение	Длина	Тип	
0	13	Байт	Имя подпрограммы (до 12 символов), дополненное двоичными 0 до 12 символов. Заканчивается двоичным 0
13	2	Слово	Смещение точки входа в подпрограмму от начала LEM модуля
15	1	Байт	Тип подпрограммы: 00h — процедура 01h — функция
16	1	Байт	Количество параметров у подпрограммы

Таблица 3

Блок описания параметра (Parameter Descriptor)

Смещение	Длина	Тип	
0	1	Байт	Тип параметра: 00h — строка 01h — слово (16 битов) 02h — двойное слово (32 бита) 03h — с плавающей точкой (FLOAT)
1	4	Двойное слово	Адрес параметра в форме SEG:OFF
5	2	Слово	Длина параметра по этому адресу

Таблица 4

Признак конца LEM модуля

Смещение	Длина	Тип	
0	1	Байт	00h

программы. Возврат из подпрограммы должен осуществляться командой дальнего возврата (FAR RET).

При передаче параметров в подпрограммы исполняющая система Clarion производит необходимые преобразования данных и помещает адрес и длину параметра в соответствующие поля блока описания параметров подпрограммы.

Если при передаче параметра в LEM модуль не требуется преобразование типов, то в LEM передается истинный адрес фактического параметра из Clarion, то есть происходит передача параметров по ссылке.

Когда в качестве фактического параметра передается структура GROUP, RECORD или TABLE, Clarion передает в LEM адрес начала этой структуры.

Таким образом, если мы хотим вернуть из LEM массив, достаточно в LEM объявить соответствующий параметр типом STRING (00), а в программе на Clarion поместить массив в структуру GROUP и передать в LEM структуру GROUP (пример 2).

LEM "Чтение абсолютного сектора логического диска ДОС" приведен в листинге 1.

Для третьего фактического параметра функции INT_25_ исполняющая система Clarion передаст в LEM адрес BUFFER, что совпадает с началом массива SECTOR, и длину, равную 512.

Для возврата значения функции программист должен установить регистры процессора следующим образом:

AL – тип возвращаемого значения:
- 00h — строка,
- 01h — слово (16 битов),
- 02h — двойное слово (32 бита),
- 03h — с плавающей точкой (FLOAT);

BX – смещение от начала LEM возвращаемого параметра;

CX – длина строки, если AL=0.

Легче всего создать требуемую структуру LEM модуля, используя язык ассемблера. Но на языке ассемблера очень часто трудоемкость программирования необходимых функций очень высока, и возникает необходимость запрограммировать их на языке более высокого уровня. При составлении LEM модулей на таких языках программирования нужно иметь ввиду, что не все языки высокого уровня (или их реализации) позволяют, в принципе, получить модули, пригодные для преобразования в формат LEM.

Создавая LEM модуль на языке высокого уровня, необходимо следовать следующим правилам.

Пример 2▼

```
DSK_NUM     SHORT           !
SEC_NUM     LONG            !LONG, потому, что нет
                            !беззнакового SHORT
ERR_COD     LONG            !LONG, потому, что нет
                            !беззнакового SHORT
BUFFER      GROUP           !Буфер для сектора
SECTOR      BYTE,DIM(512)   !Массив для сектора
.
  ERR_COD=INT_25_(DSK_NUM,SEC_NUM,BUFFER)
  !Вызов функции, реализующей чтение сектора через
  !прерывание ДОС 25h
```

Создать модуль на языке АССЕМБЛЕРА, откуда вызываются подпрограммы на языке высокого уровня. Этот модуль должен выполнять следующие функции:
- содержать заголовок LEM, блоки описания подпрограмм;
- обеспечивать увязывание соглашений о передаче параметров в LEM модуле и выбранном языке программирования и необходимых преобразований типов параметров;
- вызывать подпрограммы на выбранном языке;
- возвращать управление в Clarion.

В этом модуле надо правильно определить необходимые сегменты (их имена, типы выравнивания, классы, принадлежности к группам и тому подобное) и правильно разместить данные и код в сегментах. Для этого следует тщательно изучить руководство по используемому языку.

Часто многие конструкции языков высокого уровня и вызовы библиотечных функций не работают, если в программе не предусмотрены специальные части - пролог и эпилог (startup code), автоматически включаемые в главную программу. В нашем же случае для создания LEM главная программа не нужна. Ее роль выполняет исполняющая система Clarion. Тогда от использования конструкций и библиотечных функций, зависящих от наличия специфического для данного языка startup code, придется или отказаться, или повторить необходимые их части в своем модуле на ассемблере.

Для языка C эти startup code автоматически включаются в функцию main.

Часто исходные тексты startup code имеются в составе системы программирования.

Далее следует модуль на выбранном языке программирования с нужными подпрограммами и модуль на языке ассемблера с признаком конца LEM.

И, наконец, все сегменты из всех модулей надо разместить в EXE и BIN файле в определенном порядке.

Вся эта процедура требует кропотливого программирования и хорошего знания системных вопросов. Для облегчения создания LEM фирма Clarion Software разработала пакет LEM Maker, автоматизирующий оформление LEM модулей, написанных на Turbo C.

Теперь мы готовы рассмотреть пример LEM, содержащего две процедуры. Одна процедура включает динамик, а другая выключает его. При этом после включения динамика программа может продолжать свое выполнение, а динамик будет выдавать заданный тон до тех пор, пока его не выключить. Программирование этих процедур требует знакомства с использованием микросхемы таймера 8253 и знание языка ассемблера.

В листинге 2 приведен пример реализации этого LEM на ассемблере, а в листинге 3 - реализация LEM с использованием Microsoft C v. 5.1 .

В результате мы получили LEM beeplem.bin и beep.bin, пригодные для работы с программой-интерпретатором Processor. Для использования функций

из этих LEM в главную программу на Clarion необходимо поместить следующее описание (пример 3).

Пример 3

```
MAP
    <Другие необходимые описания>
    MODULE('BEEPLEM'),BINARY    !LEM модуль
        PROC(BEEP_OFF)          !Выключить динамик
        PROC(BEEP_ON)           !Включить динамик
```

Если мы хотим создать самостоятельно выполняемую программу, использующую этот LEM, то должны оттранслировать полученный псевдокод (PRO) программой Translator. Сделав это, мы обнаружим появление в каталоге файла с необычным расширением beeplem.boj. Этот файл вместе с другими объектными файлами и библиотеками обрабатывается компоновщиком. Что он из себя представляет и откуда берется?

По структуре это обычный объектный модуль. Создается он из двоичного файла beeplem.bin программой Translator. То есть Translator восстанавливает из двоичного позиционно независимого формата образа памяти объектный модуль. При этом имена сегментов, меток и так далее, которые были в исходном тексте, уже потеряны, и Translator создает необходимые ему новые глобальные имена. Эти имена строятся по определенным правилам из имени подпрограммы (поле имени подпрограммы из блока описания подпрограммы), которое в виде символьной строки содержится в двоичном файле LEM.

Посмотрим на выдержки из карты памяти компоновщика (рис. 2) для EXE программы, использующей

```
Start  Stop   Length Name                   Class              Group
00000H 00053H 00054H BEEPLEM_BIN_SEG       CLARION

:

 Address         Publics by Value

0000:0003       BINFNC
0000:000A       _LEM_BEEP_ON
0000:0022       _LEM_BEEP_OFF
0000:0034       _CODE_BEEP_ON
0000:004D       _CODE_BEEP_OFF
```

Рис. 2. Выдержки из карты памяти компоновщика

наш LEM. При внимательном сравнении адресов из карты памяти и листинга 1 трансляции нетрудно установить соответствие старых имен и новых.

Весь LEM помещается в сегмент ИМЯ_LEM_МОДУЛЯ_BIN_SEG.

Глобальные метки (publics) означают:

_LEM_ИМЯ_ПОДПРОГРАММЫ	–	указывает на блок описания подпрограммы.
_CODE_ИМЯ_ПОДПРОГРАММЫ	–	указывает на точку входа в подпрограмму.
BINFNC	–	указывает на некий внутренний символ среды.

Исполняющая система Clarion в EXE модуле, в отличие от процессора, не сканирует сегмент LEM, в поисках начала соответствующих таблиц, а обращается непосредственно по меткам. Поэтому для LEM в форме объектного модуля, включаемого в Clarion EXE программу, общий заголовок LEM не нужен. А вот заголовки процедур и блоки описания параметров необходимы. Исполняющая система Clarion перед вызовом подпрограммы из LEM помещает адреса и длины параметров в соответствующий блок, расположенный по адресу _LEM_ИМЯ_ПОДПРОГРАММЫ. При этом она пользуется полями типа параметра, которые заполняет разработчик LEM, для автоматического преобразования типов параметров.

Разобравшись с этим и зная соглашение о связях, назначение и названия всех групп и сегментов в Large модели памяти Turbo C (а Clarion сам написан именно в этой модели), можно создавать LEM, в значительной степени свободные от ограничений, нами разобранных, но не работающие в режиме интерпретации.

Такой LEM можно заставить работать в оверлее. Для этого необходимо блоки описания процедур и параметров, а также области для возвращаемых значений поместить в сегмент глобальных инициализированных данных (в целях обеспечения их доступности для исполняющей системы Clarion в любой момент). Код LEM надо поместить в собственный кодовый сегмент. В листинге 4 дан такой вариант LEM "Чтение абсолютного сектора логического диска".

В целом задача стыковки по управлению и данным программ, созданных в разных средах программирования, требует ясного понимания функционирования как самих этих сред (включая библиотеки времени исполнения), так и операционной системы в целом. Решение такого рода задач таит в себе много неожиданностей и трудностей и может потребовать значительного времени и усилий, которые при удачном исходе с лихвой окупятся. ■

Ориентир—1993 Журнал для пользователей

Подписной купон

Адрес _____

Ф.И.О. или название организации _____

_____

Вышлите подписной купон по адресу: 117419, Москва, а/я 765.

Листинг 1. LEM "Чтение абсолютного сектора логического диска"

BIN файл создавать, как описано выше.

```
Lem     SEGMENT  BYTE
;
        ASSUME   CS:Lem,DS:Lem
;
;Общий заголовок LEM модуля.
;
        DB  'BIO'               ;Сигнатура LEM модуля
        DD  0                   ;Резерв
        DW  OFFSET LemEnd       ;Длина LEM модуля
        DB  1                   ;Одна функция в этом модуле
;
;Заголовок с описанием функции LEM модуля
;
;Чтение абсолютного сектора логического диска ДОС.
;
        DB  'INT_25_',0,0,0,0,0,0;Имя функции
        DW  OFFSET INT_25_      ;Смещение точки входа в процедуру
        DB  1                   ;Тип: функция
        DB  3                   ;Три параметра
;Описание параметров функции
;Номер дисковода
        DB  1                   ;16-разрядное слово
DskNum  DD  0                   ;Адрес параметра в формате
                                ;смещение:сегмент
        DW  0                   ;Длина параметра
;Логический номер сектора
        DB  2                   ;32-разрядное слово, потому что нет
                                ;беззнакового SHORT
SecNum  DD  0                   ;Адрес параметра в формате
                                ;смещение:сегмент
        DW  0                   ;Длина параметра
;Буфер для чтения
        DB  0                   ;Строка, чтобы не производилось
                                ;преобразование типов
                                ;При вызове указывается метка
                                ;оператора GROUP
SecBuf  DD  0                   ;Адрес параметра в формате
                                ;смещение:сегмент
        DW  0                   ;Длина параметра
;Конец описания параметров функции.
;
;Функция чтения сектора. Читает всегда один сектор.
;Не проверяет:  -  соответствие размера буфера размеру сектора;
;               -  возможность пересылки сектора по адресу, указаниому в
;                  блоке описания параметра, то есть
;                  смещение_буфера+длина_сектора не должно переходить
                   ;через границу сегмента
;
;Возвращает      0, если все OK,
;                код ошибки, как его дает прерывание ДОС.
;
RetVal  DW  0                   ;Возвращаемое значение
        DW  0
;
INT_25_ PROC  FAR
;Подготовить регистры для вызова int25
        mov cx,1                ;читаем один сектор
        les  bx,DskNum          ;в ES:BX адрес номера дисковода
        mov al,BYTE PTR es:[bx] ;младший байт - номер дисковода
        les  bx,SecNum          ;в ES:BX адрес номера сектора
        mov dx,WORD PTR es:[bx];младшее слово - логический номер сектора
        les  bx,SecBuf          ;в ES:BX адрес буфера
; Просто используем стек исполняющей системы Clarion, надеясь, что он
; не переполнится
        push ds                 ;запомним текущий DS
        push es                 ;подготовим адрес буфера
        pop  ds                 ;в форме DS:BX
;Вызов прерывания ДОС
        int  25h
        pop  dx                 ;int25 оставляет лишнее слово в стеке
        pop  ds                 ;восстановим старое значение DS
        jc   Error              ;Если ошибка
        mov  RetVal,0           ;Все в порядке, возвращаем 0
        jmp  Done
Error:  mov  RetVal,ax          ;Если ошибка, то возвращаем ее код
Done:   mov  al,2               ;функция возвращает двойное слово
        mov  bx,OFFSET RetVal   ;смещение возвращаемого значения
        ret
INT_25_ ENDP
;
LemEnd  DW  0                   ;Признак конца LEM модуля
;
Lem     ENDS                    ;Конец сегмента
        END
```

Листинг 2. Реализация LEM BEEP на ассемблере

Последовательность создания LEM:
MASM beeplem;
результат: beeplem.obj.
Компоновка:
LINK beeplem;
результат: beeplem.exe.
Компоновщик Microsoft Link выдает обычно сообщения:
"LINK : warning L4021: no stack segment" —
"Предупреждение. Нет сегмента стека."
и
"LINK : warning L4038: program has no starting address" —
"Предупреждение. У программы отсутствует стартовый адрес"
Преобразование в двоичный позиционно независимый формат образа памяти:
EXE2BIN beeplem.exe beeplem.bin
Результат: beeplem.bin

```
;================================================
0000                    Lem  SEGMENT BYTE
                        ASSUME    CS:Lem,DS:Lem
                        ;Общий заголовок LEM модуля
0000  42 49 4F                  DB   'BIO'   ;Сигнатура LEM модуля
0003  00000000                  DD   0       ;Резерв
0007  0054 R                    DW   OFFSET LemEnd ;Длина LEM
                                             ;модуля
0009  02                        DB   2       ;Две процедуры или
                                             ;функции в этом модуле
                        ;Заголовки с описанием функций и процедур LEM
                        ;модуля
                        ;Включить звук
000A  42 45 45 50 5F 4F 4E      DB   'BEEP_ON',0,0,0,0,0,0  ;Имя
      00 00 00 00 00 00         ;процедуры, дополненное до длины
                                ;13 нулями
0017  0034 R                    DW   OFFSET Beep_On ;Смещение
                                             ;точки входа в процедуру
0019  00                        DB   0       ;Тип: процедура
001A  01                        DB   1       ;Один параметр
                        ;Описание параметров процедуры
001B  01                        DB   1       ;Параметр—
                                             ;16-разрядное слово
001C  00000000          Freq    DD   0       ;Адрес параметра в
                                ;формате смещение:сегмент
                                ;помещается исполняющей
                                ;системой Clarion перед
                                ;вызовом процедуры
0020  0000                      DW   0       ;Длина параметра
                        ;Выключить звук
0022  42 45 45 50 5F 4F 46      DB   'BEEP_OFF',0,0,0,0,0   ;Имя
      46 00 00 00 00 00         ;процедуры, дополненное
                                ;до длины13 нулями
002F  004D R                    DW   OFFSET Beep_Off ;Смещение
                                             ;точки входа в процедуру
0031  00                        DB   0       ;Тип: процедура
0032  00                        DB   0       ;Без параметров
                        ;Данные для процедур
= 0061                  SpeakerPort  EQU  061H  ;Порт включения динамика
= 0043                  TimerControl EQU  043H  ;Командный порт таймера
= 0042                  TimerCount2  EQU  042H  ;Порт счетчика канал 2

0033  B6                Mode db  10110110B ;Командное слово таймера 8253
                        ;        ||||||||
                        ;        ||||||->  Двоичный счет
                        ;        ||||--->  Меандр
                        ;        ||----->  Два байта счетчика
                        ;        ------->  Канал #2 (динамик)
                        ;Включить звук
```

Листинг 2 (продолжение)

```
0034                    Beep_On     PROC FAR
0034  A0 0033 R             mov     al,mode         ;Установить режим таймера
0037  E6 43                 out     TimerControl,al
0039  C4 1E 001C R          les     bx,Freq         ;Загрузить в ES:BX
                                                    ;адрес параметра
003D  26: 8B 07             mov     ax,WORD PTR ES:[BX] ;Загрузить в
                                                    ;AX параметр
0040  E6 42                 out     TimerCount2,al  ;Записать в
                                                    ;регистр таймера
                                                    ;коэффициент деления
0042  86 C4                 xchg    al,ah           ;Младший и старший байты
0044  E6 42                 out     TimerCount2,al
0046  E4 61                 in      al,SpeakerPort  ;Прочитать
                                                    ;регистр управления
                                                    ;динамиком
0048  0C 03                 or      al,00000011b    ;Установить
                                                    ;два последних бита
004A  E6 61                 out     SpeakerPort,al  ;Включить
                                                    ;динамик
004C  CB                    ret                     ;Возврат
004D                    Beep_On     ENDP
      ;
      ;Выключить звук
      ;
004D                    Beep_Off    PROC    FAR
004D  E4 61                 in      al,SpeakerPort  ;Прочитать
                                                    ;регистр управления
                                                    ;динамиком
004F  24 FC                 and     al,11111100b    ;Сбросить два
                                                    ;последних бита
0051  E6 61                 out     SpeakerPort,al  ;Выключить
                                                    ;динамик
0053  CB                    ret     ;Возврат
0054                    Beep_Off    ENDP
0054  0000               LemEnd      DW  0           ;Признак конца LEM
                                                    ;модуля.
0056  Lem                ENDS        ;Конец сегмента
                          END
```

Листинг 3. Реализация LEM на языке C

Последовательность создания LEM:
MASM beep1;
MASM beep3;
Для примера выбираем компактную модель памяти. Она годится для преобразования в двоичный позиционно независимый формат памяти, так как код в ней не содержит межсегментных переходов и вызовов.
CL /c /AC beep2.c
Результат: beep1.obj beep2.obj и beep3.obj
LINK beep1+beep2+beep3,beep,beep,CLIBCE/NOD
CLIBCE.LIB - объединенная библиотека (combined library) компактной модели памяти.
Результат: beep.exe
EXE2BIN beep.exe beep.bin
Результат: beep.bin
beep1.asm:

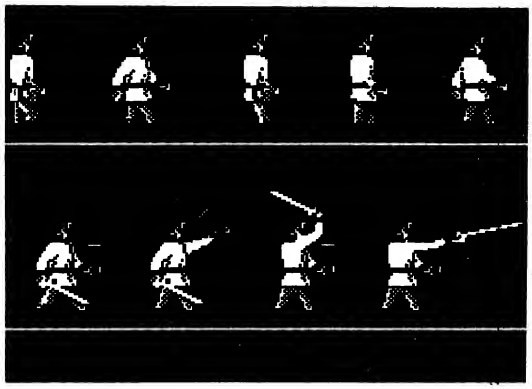
```
;
;
;=========================================
;Наименование сегментов в COMPACT модели памяти
;Сегменты, создаваемые компилятором Microsoft C
;=========================================
; Код
_TEXT   SEGMENT BYTE  PUBLIC  'CODE'
_TEXT   ENDS
; Инициализированные данные
_DATA   SEGMENT WORD PUBLIC  'DATA'
_DATA   ENDS
; Константы - строки и плавающая точка
CONST   SEGMENT WORD PUBLIC  'CONST'
CONST   ENDS
; Неинициализированные данные
_BSS    SEGMENT WORD PUBLIC  'BSS'
_BSS    ENDS
;=========================================
;Сегмент с признаком конца LEM - наш собственный.
;=========================================
LemEnd  SEGMENT WORD PUBLIC'LEMEND'
LemEnd  ENDS
;=========================================
DGROUP  GROUP   _TEXT,_DATA,_BSS,CONST,LemEnd
;=========================================
; В точности повторяет часть Microsoft C startup code (CRT0.asm):
public      __acrtused          ; trick to force in startup
            __acrtused = 9876h  ; funny value not easily matched in SYMDEB
;
; Транслятор Microsoft C включает ссылку на этот символ в каждый модуль.
; Цель включения транслятором ссылки на этот глобальный символ
; в каждый модуль - заставить компоновщик подстыковать startup code и
; определить затем отсутствие функции main.
; Мы включаем определение этого символа к себе в программу, чтобы
; избавиться от необходимости иметь функцию main. ВНИМАНИЕ ! ! !
; Без функции main очень многие библиотечные функции не компонуются
; и не могут работать.
;
;=========================================
;
_TEXT   SEGMENT BYTE  PUBLIC  'CODE'
;
        ASSUME  CS:_TEXT,DS:DGROUP
;
        EXTRN   _BEEP_ON_C:NEAR;Внешние C функции
        EXTRN   _BEEP_OFF_C:NEAR;C добавляют к внешнему имени
                                ;символ '_'
        EXTRN   LemEndMark:BYTE;Признак конца LEM
;Общий заголовок LEM модуля.
;
        DB      'BIO'   ;Сигнатура LEM модуля
        DD      0 ;Резерв
        DW      OFFSET LemEndMark;Длина LEM модуля
        DB      2               ;Две процедуры или функции в этом модуле
;
;Заголовки с описанием функций и процедур LEM модуля
;
;Включить звук
        DB      'BEEP_ON',0,0,0,0,0,0;Имя процедуры, дополненное до
                                ;длины 13 нулями
        DW      OFFSET Beep_On;Смещение точки входа в процедуру
        DB      0               ;Тип: процедура
        DB      1               ;Один параметр
;Описание параметров процедуры
        DB      1               ;Параметр - 16-разрядное слово
Freq    DD      0               ;Адрес параметра в формате
; смещение:сегмент помещается исполняющей системой Clarion перед
; вызовом процедуры
        DW      0               ;Длина параметра
;Выключить звук
        DB      'BEEP_OFF',0,0,0,0,0;Имя процедуры, дополненное до
                                ;длины 13 нулями
        DW      OFFSET Beep_Off;Смещение точки входа в процедуру
        DB      0               ;Тип: процедура
        DB      0               ;Без параметров
;
;Включить звук
;
Beep_On PROC    FAR
        les     bx,Freq         ;Загрузить в ES:BX адрес параметра
; Просто используем стек исполняющей системы Clarion, надеясь, что он
; не переполнится
        push    WORD PTR ES:[BX];Загрузить в стек параметр
        call    _BEEP_ON_C      ;Вызвать подпрограмму на C
        add     sp,2            ;Восстановить стек
        ret                     ;Возврат
Beep_On ENDP
;
;Выключить звук
;
Beep_Off PROC   FAR
        call    _BEEP_OFF_C     ;Вызвать подпрограмму на C
        ret                     ;Возврат
Beep_Off ENDP
_TEXT   ENDS ;Конец сегмента
        END
```

```
beep2.c:
#include <conio>.h
/*=========================================*/
/* Специфично для реализации Microsoft C   */
/* Для исключения ссылок на STARTUP CODE   */
/*        выключить проверку стека         */
#pragma check_stack(off)
/*=========================================*/
void BEEP_ON_C( unsigned frequency )
{
        int   control;
              /* Подготовить таймер, послав 10111100 в порт 43. */
              outp( 0x43, 0xb6 );
              /* Сохранить управляющий байт динамика */
              control = inp( 0x61 );
              /* Делитель частоты записать побайтно в таймер. */
              outp( 0x42, (char)frequency );
              outp( 0x42, (char)(frequency > 8) );
              /* Включить динамик. */
              outp( 0x61, control | 0x3 );
}
void BEEP_OFF_C(void)
{
        int   control;
              /* Сохранить управляющий байт динамика */
              control = inp( 0x61 );
              /* Выключить динамик. */
              outp( 0x61, control & (~0x3) );
}
beep3.asm:
;
;Сегмент с признаком конца LEM
;
LemEnd        SEGMENT  WORD PUBLIC'LEMEND'
;
              PUBLIC     LemEndMark
;
LemEndMark  DB         0
;
LemEnd        ENDS
              END                                  ■
```

Листинг 4. LEM "Чтение абсолютного сектора логического диска" в форме объектного модуля, включаемого в Clarion EXE программу

Трансляция: MASM int25lem.asm;
Компоновка: модуль int25lem.obj включить в список модулей для компоновки.
Если файл автоответов для копоновщика создавался Clarion, то удалить из списка модулей int25lem.boj. Он помещается туда, так как функции из LEM должны быть описаны в структуре MAP.

```
;
;=============================================
;Наименование сегментов в LARGE модели памяти
;=============================================
;
;Кодовый сегмент LEM
;
INT_25__TEXT SEGMENT  PUBLIC  BYTE'CODE'
INT_25__TEXT ENDS
;
;=============================================
;Глобальные инициализированные данные
;
_DATA         SEGMENT  PUBLIC  BYTE'DATA'
_DATA         ENDS
;
;Глобальные неинициализированные данные
;
_BSS          SEGMENT  PUBLIC  BYTE'BSS'
_BSS          ENDS
;
;Глобальные данные собраны в одну группу
;
DGROUP        GROUP    _DATA,_BSS
;=============================================
;
              ASSUME   CS:INT_25__TEXT,DS:DGROUP
;
_DATA         SEGMENT  PUBLIC  BYTE'DATA'
;
              PUBLIC   _LEM_INT_25_
_LEM_INT_25_  LABEL    BYTE
;
              DB  '    INT_25_',0,0,0,0,0,0;Может быть любое значение,
              DW       OFFSET _CODE_INT_25_ ;так как в EXE программе
                       ;содержимое этой области не используется
                       ;исполняющей системой Clarion
              DB       1          ;Тип: функция
              DB       3          ;Три параметра
;
;Описание параметров функции
;
;Номер дисковода
              DB       1          ;16-разрядное слово
DskNum        DD       0          ;Адрес параметра в формате
                                  ;смещение:сегмент
              DW       0          ;Длина параметра
;Логический номер сектора
              DB       2          ;32-разрядное слово
SecNum        DD       0          ;Адрес параметра в формате
                                  ;смещение:сегмент
              DW       0          ;Длина параметра
;Буфер для чтения
              DB       0          ;Строка, чтобы не производилось
                                  ;преобразование типов
                                  ;При вызове указывается метка
                                  ;оператора GROUP.
SecBuf        DD       0          ;Адрес параметра в формате
                                  ;смещение:сегмент
BufLen        DW       0          ;Длина параметра
;
;Конец описания параметров функции
;
RetVal        DW       0          ;Возвращаемое значение
              DW       0
;
_DATA         ENDS
;
INT_25__TEXT SEGMENT  PUBLIC  BYTE   'CODE'
;
              PUBLIC   _CODE_INT_25_
_CODE_INT_25_  PROC      FAR
;
;Подготовить регистры для вызова int25
              mov      cx,1       ;читаем один сектор
              les      bx,DskNum;в ES:BX адрес номера дисковода
              mov      al,BYTE PTR es:[bx] ;младший байт —
                       ;номер дисковода
              les      bx,SecNum;в ES:BX адрес номера сектора
              mov      dx,WORD PTR es:[bx];младшее слово —
                       ;логический номер сектора
              les      bx,SecBuf;в ES:BX адрес буфера
              push     ds         ;запомним текущий DS
              push     es         ;подготовим адрес буфера
              pop      ds         ;в форме DS:BX
;Вызов прерывания ДОС
              int      25h
              pop      dx         ;int25 оставляет лишнее слово в стеке
              pop      ds         ;восстановим старое значение DS
              jc       Error      ;Если ошибка
              mov      RetVal,0   ;Все в порядке, возвращаем 0
              jmp      Done
;
Error:        mov      RetVal,ax ;Если ошибка, то возвращаем ее код
;
Done:         mov      al,2       ;функция возвращает двойное слово
              mov      bx, OFFSET RetVal;смещение возвращаемого
                       ;значения
              ret
;
_CODE_INT_25_  ENDP
;
INT_25__TEXT    ENDS
;
                 END                                ■
```



Adventure Games: эволюция жанра

Часть 2

Развитие средств вычислительной техники в начале 80-х годов значительно повлияло на дальнейшую эволюцию игр жанра ADVENTURE. В первую очередь это относится к появлению средств графической поддержки. В этой статье мы остановимся на том, как были использованы графические возможности персональных ЭВМ.

Можно выделить два основных направления, по которым пошло развитие игр после начала широкого внедрения графики. Во-первых, графика стала применяться в качестве иллюстративного материала, поддерживая или заменяя традиционные текстовые описания. Во-вторых, графика стала неотъемлемым элементом системы управления игрой, что постепенно привело к отказу от ввода команд с клавиатуры.

Графика как иллюстративный материал

Первым шагом в этом направлении стало появление статических фоновых экранов, как, например, в практически чисто текстовой программе "The Hobbit" фирмы MELBOURNE HOUSE. Ранняя иллюстративная графика имела векторный характер; черно-белое изображение рассчитывалось по заданному алгоритму, а цвет обеспечивали процедуры "заливки" замкнутых контуров. Конечно, при таком подходе изображения имели определенный примитивизм, но были и удачные решения. Так, например, в той же программе "The Hobbit" были воспроизведены в основном рисунки самого Дж.Р.Р.Толкиена к его книге, и программа приобрела очаровательный колорит.

К этому периоду относится начало развернутой дискуссии на страницах зарубежных компьютерных журналов о том, нужна ли графика в адвентюрных играх. Сторонники графики справедливо видели в ней новые перспективы развития жанра.

Противники не менее справедливо указывали на то, что главным в жанре является диалог и введение графики уводит жанр в сторону. Конечно, дискуссия велась не абстрактно, а на конкретных примерах коммерческих программ и, как это и должно быть, появление новых удачных продуктов поставило в ней окончательную точку. Оказалось, что истина в разумном компромиссе. Вся проблема - в балансе графического и текстового материала. Оптимальный путь не в отказе от текста, а в дополнении его, если это не приводит к снижению содержательности и интерактивности из-за ограниченного размера оперативной памяти и дискового пространства или при низкой производительности процессора.

Не удивительно, что дальнейшим шагом стало появление программ, в которых графика стала не просто иллюстрацией. На экранах стали прятать элементы, необходимые для успешного прохождения игры, но не объявленные в текстовом сопровождении. Примеров можно не приводить: практически все современные игры применяют этот прием.

Период ранней векторной графики длился очень недолго и вскоре с появлением жестких дисков сменился следующим - введением растровой графики высокого разрешения, а вслед за тем и динамической графики.

В этот момент произошло окончательное разделение двух классов: Role-Playing Games и Adventure Games. Role-Playing Games эволюционировали к емким игровым полям размером в несколько тысяч локаций и к лабиринтам относительно простой структуры, где игровое поле изображается либо видом сверху, либо как "слепой лабиринт", когда персонаж смотрит вперед и видит все, что не закрыто стенами на некотором расстоянии впереди себя. В таких играх основным элементом является сражение с превосходящими силами противника, и этот мотив подавляет логический элемент. Данный класс

игр заслуживает отдельной статьи, и мы вернемся к нему в последующих номерах журнала.

Adventure Games пошли по другому пути. Первой пробой стала программа "King's Quest I" фирмы SIERRA ON-LINE. Своим появлением эта игра произвела настоящую сенсацию, было продано около миллиона экземпляров. Она впервые (на IBM совместимых машинах) ввела в адвентюру элементы мультипликации. В основном это относилось к главному персонажу, управление которым осуществлялось теперь при помощи курсорных клавиш. Кроме того, имелось еще несколько подвижных персонажей, встреча с которыми была либо полезной и помогала главному герою, либо грозила ему неприятностями и даже гибелью. Введение нескольких планов изображений, за которыми мог двигаться персонаж, придало графике зрительную объемность.

Если смотреть на эту игру, а также на последовавшую за ней "King's Quest II" с современной точки зрения, то видны некоторые издержки развития. В первой игре сюжетная линия довольно слаба, и игра представляет собой практически "сбор сокровищ", весьма характерный для ранних адвентюр. "Сокровища" надо собирать и во второй игре, хотя здесь уже виден четко выраженный сюжет. Кроме того, от текстовых игр унаследовано и соответствие экран-локация, то есть чтобы взять или использовать какой-либо предмет, достаточно выдать команду, находясь в любой точке соответствующего экрана. Это удобно с точки зрения управления игрой, но, с другой стороны, выполнение такого действия выглядит несколько неестественным. Далее мы увидим, как разрешилась эта проблема.

Количество локаций в этих играх относительно невелико - порядка сотни. Такое число de-facto стало стандартным в современных адвентюрах, поскольку позволяет реализовать достаточно интересный и продолжительный сюжет и в то же время в разумных пределах сохраняется время, требуемое на прохождение игры, и размер занимаемого ею дискового пространства. Правда, в последнее время четко видна тенденция к росту второго параметра. Игры все чаще достигают размеров до 10 мегабайтов и более.

Увеличение игрового пространства достигается часто приемом типа "путешествие по пустыне", как в "King's Quest V" и "Hero Quest II", что требует введения необычной единицы измерения расстояния под названием "экран". В "Hero Quest II" она используется даже явным образом, когда герою подсказывают, как найти оазис в пустыне. Применяется для этих целей также и "слепой лабиринт" ("King's Quest V"). Остроумным приемом является "путешествие по карте", использованное в игре "The Secret of Monkey Island" фирмы LUCASFILM GAMES, а также в "Hero Quest III". Оригинальный прием использован в игре "Mean Streets" фирмы ACCESS SOFTWARE. Все локации задаются координатами, причем герой узнает эти координаты по ходу игры.

Относительно небольшое количество локаций приводит к тому, что повышается их насыщенность. В современных адвентюрах практически нет "проходных" локаций - везде что-то можно и нужно сделать. Некоторые локации бывают очень насыщенными. Так, например, в спальне колдуна в "King's Quest III" нужно достать магическое зеркало из ящика, найти ключ и взять в шкафу магическую карту. Там может находиться кошка, у которой надо взять немного шерсти, и может спать колдун, которого можно (но не нужно) разбудить.

Графика как элемент системы управления

Введение динамической графики имело главной целью анимацию иллюстративного материала и привело к неожиданным последствиям. Частично сбылись предсказания противников графики: игры стали эволюционировать в направлении аркадного (ARCADY) жанра. Во-первых, выделился совершенно независимый жанр аркадных адвентюр (ARCADE ADVENTURE), а во-вторых, многие адвентюрные программы стали практиковать так называемые "аркадные вставки".

Переходной программой к аркадным адвентюрам считается программа "Atic Attack" фирмы ULTIMATE. Это направление характеризуется, прежде всего, системой управления, ранее принятой для аркадных игр, - 5-6 клавиш при полном отказе от текстового ввода. Все управление героем на экране выполняется с помощью четырех курсорных клавиш и одной-двух функциональных. От адвентюрных игр этот жанр унаследовал использование объектов, распределенных по игровому пространству для выполнения оперативных задач. Основным же отличием от классических адвентюр стал полный отказ от диалога как элемента системы управления игрой. Первой программой, открывшей жанр, стала игра "Knight Lore" фирмы ULTIMATE. Эта же игра ввела в действие своеобразный "стандарт" трехмерного представления игрового пространства, реализуемый в настоящее время во многих играх (например, "Lure of the Temptress" фирмы VIRGIN GAMES).

Жанр аркадных адвентюр стал любимым жанром миллионов пользователей персональных компьютеров у нас в стране. Особую популярность ему придает тот факт, что на уровне 8-разрядных бытовых машин ассортимент этих программ измеряется многими сотнями, но мы вернемся к "чистым" адвентюрам.

Второе поколение игр (например "King's Quest III", "Leisure Suit Larry I", "Space Quest I" и "Police Quest I") использовало режим EGA 320x200x16. Связь экран-локация в них уже не была столь выраженной. Элементы изображения получили независимую значимость, то есть чтобы открыть шкаф, персонаж должен подойти именно к шкафу, а не находиться в произвольном месте соответствующего экрана. Такой способ управления иногда вызывал не-

которые трудности - мнения программы и пользователя о том, где должен стоять персонаж для доступа к объекту, не всегда совпадали.

В третьем поколении еще более повысилось качество графических изображений и появилась возможность использования в системе управления манипулятора типа "мышь". Для перемещения персонажа в заданную точку экрана стало достаточно нажать в этом месте левую кнопку "мыши" - персонаж сам найдет маршрут движения. Элементы экрана окончательно приобрели самостоятельное значение - нажатием правой кнопки "мыши" вызывается описание соответствующего объекта. Одновременно было реализовано для большинства локаций и автоматическое движение персонажа по экрану между двумя позициями или для доступа к объекту. Стало ясно, что текстовый ввод команд начал устаревать.

Фирмы, производящие адвентюрные игры, отреагировали на это, и большинство программ, выпущенных в последние три года, имеют небольшие меню различного рода. Абсолютное большинство действий, выполняемых персонажем при прохождении игры, достаточно стандартны: взять-положить (TAKE/-DROP) предмет, изучить (EXAMINE), передвинуть (MOVE) объект, открыть-закрыть (OPEN/CLOSE) что-либо, включить-выключить (TURN ON/TURN OFF) свет/прибор, поговорить (TALK TO) с другим персонажем. Нестандартные действия, как правило, требуют применения какого-либо инструмента или объекта и могут исполняться по команде "использовать" (USE). Таким образом, пользователь имеет в своем распоряжении меню, состоящее из 5-15 пунктов. Из этого меню выбирается глагол. В качестве дополнения из меню второго уровня выбираются объекты, имеющиеся у персонажа, или те, к которым он имеет доступ на текущем экране. Так формируется команда. Фирма LUCASFILM GAMES реализовала даже автоматическое определение на экране объектов, которые могут иметь значение для игры. Чтобы ввести название объекта в команду, достаточно указать на него курсором "мыши".

При общении с другим персонажем текст фразы либо формируется автоматически (большинство игр фирмы SIERRA ON-LINE), либо предлагается в виде меню (большинство игр фирмы LUCASFILM GAMES). Иногда меню представляет не готовые фразы, а перечень возможных намерений: угроза, обман, просьба о помощи, запрос информации и так далее - "Countdown" фирмы ACCESS SOFTWARE.

Другой, не менее часто применяющийся способ управления программой состоит в использовании пиктограмм. Впервые такой прием был применен в адвентюрном детективе "The Fourth Protocole", подготовленном фирмой CENTURY по одноименному роману известного английского писателя Ф.Форсайта. Суть этого подхода состоит в том, что каждому возможному действию на экране соответствует своя пиктограмма. Совокупность пиктограмм образует систему меню. Желая позвонить кому-либо, играющий выбирает пиктограмму с изображением телефона. Выбрать абонента можно, указав на пиктограмму записной книжки, которая развернется в виде записей, образующих меню нижнего уровня. В последние годы этот способ управления наиболее активно поддерживается фирмой SIERRA ON-LINE.

Как мы указали ранее, развитие техники программирования и переход к динамической графике позволили многим фирмам внедрить в свои программы так называемые "аркадные вставки". Особенно увлекается такими приемами та же фирма SIERRA ON-LINE. Нередко по ходу игры встречаются ситуации, в которых от скорости действий играющего зависит "жизнь" героя или успех завершения игры. Возможные варианты: поединок, погоня, имитация игрового автомата, логической или азартной игры. Фирма, как правило, жестоко наказывает пассивного игрока. Печальная гибель в пасти подводного чудовища ждет героя в программе "Space Qwest IV", если он не успеет освободить себя от пут и бросить в чудище баллон со сжатым газом. Впрочем, для тех, кто не любит подобные переживания, фирма оставляет возможность уменьшения скорости игры и вводит дополнительно в качестве настроечного параметр "аркадности", который тоже можно понизить.

Несколько мягче с героем обходится в своих программах фирма LUCASFILM GAMES - здесь от него не требуется особой поспешности действий, но если логика этих действий понята неправильно, то никакая поспешность не поможет. Открыть тюремную решетку в программе "The Secret of Monkey Island" можно, только расплавив замок с помощью грога (излюбленный пиратский напиток). Но донести кружку грога до тюрьмы Вы не сумеете, если не сообразите, что надо взять пять кружек и, по мере того, как очередная будет плавиться, грог следует переливать в новую кружку.

* * *

Закончить свой обзор эволюции игр жанра ADVENTURE мы хотели бы теми же словами, с которых начали. Эти программы являются отличным средством обучения диалогу с компьютером, мощным средством поддержки для изучающих английский язык и оказывают влияние на общую компьютерную грамотность. Программисты-профессионалы найдут в этих играх интересные приемы организации интерфейса "человек-компьютер", приобретут необходимое творческое вдохновение. В ближайшие годы можно предвидеть необычайно высокий интерес к играм этого жанра в нашей стране, и мы заинтересованы в расширении уровня информированности наших читателей и в установлении обратной связи с ними.

Все упомянутые в статье и многие другие адвентюрные программы есть в распоряжении "ИНФОРКОМа", и мы готовы продолжить диалог с заинтересованными читателями, с радостью примем крити-

ческие замечания, дополнения, поделимся полезными советами.

Для нас игры этого жанра имеют реальный прикладной интерес. "ИНФОРКОМ" ведет целенаправленное исследование, посвященное вопросам применения концепции адвентюрных игр в обучающем программировании. С одним из результатов этого исследования можете познакомиться и Вы. Это наша обучающая программа по истории — "Смутное Время". Программа посвящена труднейшему периоду Российского Государства — 1598-1612 г.г. В рамках адвентюрной игры учащийся не только подробнейшим образом изучит исторический материал, недостаточно освещенный в школьных учебниках, но и исследует возможные, но нереализовавшиеся в силу каких-то исторических причин пути развития России, познакомится со многими историческими деятелями.

Напоминаем наш адрес:
121019, Москва Г-19, а/я 16,
"ИНФОРКОМ". ■

Phyton (R)

Фирма "Ф Й Т О Н", Москва
тел/факс 481-05-83, тел.211-83-93

Программаторы РПЗУ серии 573 (2716...274000), 8748, 8751; дисассемблер 8048/51, 8080/85, 8088/86, Z-80

ВНУТРИСХЕМНЫЕ ЭМУЛЯТОРЫ реального времени с поддержкой символьной отладки на ассемблере и языках высокого уровня. Эмуляция КМОП-версии, мощный многооконный интерфейс пользователя, аппаратная трассировка.

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ДЛЯ 1816/30/35/50 ВЕ 35...39...49...31...51... (Intel 80[illegible]...8051..8044..8096), Zilog Z-80

Полные руководства по большинству цифровых/ аналоговых ИМС, по i8096.

Интегрированные пакеты программ в TURBO-стиле для языков C-51, ASM-51, PL/M-51 ASM-96, ASM-48, ASM-80:
■ редактор текста ■ отладчик
■ компилятор ■ симулятор

КОНТРОЛЛЕР-конструктор для отработки схем и программ систем с 1816ВЕ31. Загрузка и запуск программ с IBM PC без прожига ПЗУ.

Портативный МИКРОМАНОМЕТР-РАСУОДОМЕР с трубкой Пито и со сложной обработкой профилей скорости для контроля вентиляции, экологических тестов и т.д.
Предел - 250мм.вод.ст. Точность - 0,1+0,01*P мм.вод.ст.

ULTER WEST

9 РАБОЧИХ МЕСТ НА ОДНОМ 286 - ЭТО РЕАЛЬНОСТЬ

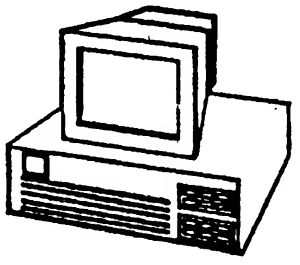
Наши программы работают в многотерминальном режиме
* ТЕКСТОВЫЙ РЕДАКТОР, идеальное средство для подготовки текстов;
* РЕДАКТОР ФАЙЛОВ СУБД PARADOX, от средства массированного ввода данных в Вашу БД до самостоятельной информационно-поисковой системы;
* ОПЕРАЦИОННЫЙ ДЕНЬ БАНКА, полный информационный цикл банка от управляющего до операционистки.

ВСЕ ЭТО РЕАЛИЗОВАНО НА PC_ MULTI

PC _ MULTI - это инструментальное средство для создания многотерминальных систем
* до 9 рабочих мест на IBM PC 286, используя видеотерминалы типа VT52/100;
* среда MS-DOS, язык разработки Turbo C;
* СУБД совместимая с PARADOX и dBase.

Ulter_ LAN - ЛОКАЛЬНАЯ СЕТЬ

*недорогая и простая в обращении;
* различные архитектуры соединений;
* файловые операции для любых программ;
* эффективное расширение числа рабочих мест станций сети фирмы Novell Net Ware.



ФИРМА "АЛЬТЕР—ВЕСТ"

продает компьютеры производства США стандартной конфигурации и по заказу, другую оргтехнику.

АДРЕС: 125140, Москва, Ленинградский проспект15
ТЕЛЕФОН: (095) 251-2946, 213-4416
ФАКС: (095) 348-9886
E-mail igor@ulter. pccentre. msc. su.

Ком–СТЕЛС — телекоммуникации в России

Эта статья адресована тем читателям, перед которыми стоит задача обеспечения информационного обмена. Если технология создания локальных вычислительных сетей уже достаточно хорошо освещена, то вопросы создания распределенных вычислительных сетей и организации электронной почты по-прежнему носят характер "общих рекомендаций" и завершаются советом "вступайте в **RELCOM**". Но всегда ли нужно подключаться к сети "всемирного" масштаба, если перед Вами стоит, например, задача организации обмена данными с филиалами, размещенными в других городах или даже на соседней улице?

А почему бы не организовать собственную сеть передачи данных по телефонному каналу? Именно по этому пути пошло малое предприятие "Ком-СТЕЛС", разработавшее телекоммуникационную сеть **STEALTH**. Сеть ориентирована на конечного пользователя, обладающего ограниченными познаниями в области вычислительной техники и телекоммуникаций. В результате принять или отправить почту Вы можете через десять минут после начала знакомства с программным комплексом. Текущая версия (**STEALTH 2.1**) работает со стандартными (**HAYES** совместимыми) модемами по обычным телефонным (невыделенным) каналам связи. (Забегая вперед, отметим, что новая версия **STEALTH 3.0**, анонсируемая на выставке "Биржевые и банковские системы", поддерживает работу и с большинством отечественных разработок типа **Lexand, Мастак, ФизТех** и так далее).

Система **STEALTH** обеспечивает выполнение следующих основных операций:

- ведение многоуровневого телефонного справочника;
- формирование задания для передачи и установка его в очередь заданий (с возможностью установки времени начала "дозвона" и приоритета задания — обычное или срочное);
- автоматический выбор задания из очереди, "дозвон" и передача информации с использованием оригинального помехоустойчивого протокола связи **SMODEM** (кроме того, для совместимости со стандартными телекоммуникационными пакетами поддерживается и **XMODEM**);
- автоматический дозвон по заданному номеру для телефонного разговора;
- прием информации, поступающей от других ЭВМ, а также передача данных осуществляются в *фоновом режиме*;
- возможность шифрования передаваемой в канал информации.

Перечисленные выше возможности сети **STEALTH** отличаются от зарубежных аналогов (к сожалению, о сопоставимых по уровню отечественных разработках говорить пока не приходится) прежде всего ориентацией на отечественного пользователя (русскоязычный интерфейс, управление файлами в стиле *Norton Commander* и так далее). Но есть и ряд особенностей, которые заинтересуют программистов:

- диагностика наличия и работоспособности модема и телефонной линии (поддержка портов COM1..COM4);
- управление режимом работы драйвера модема из прикладных программ (с использованием мультиплексного вектора **2F**);
- формирование заданий и установка их в очередь из прикладных программ;
- резидентный драйвер управления модемом использует всего 11 КБайтов оперативной памяти ЭВМ;
- организация удаленного управления компьютером по телефонному каналу (вплоть до использования обычного *Norton Commander* на другом конце линии!); при этом на ведомой ЭВМ устанавливается драйвер Remote.exe, занимающий только 25 КБайтов ОЗУ;
- сценарный язык управления передачей электронной почты (пример 1).

Перечисленные возможности позволяют сопоставлять систему **STEALTH** с такими известными продуктами, как *Telix, Procomm, ComIt*. Конечно, разработка фирмы "Ком-СТЕЛС" не предназначена для "выкачивания" GIF файлов из BBS, но ведь пакет и ориентирован, как уже упоминалось, на создание своей собственной, частной сети передачи данных (использующей при необходимости криптографическое закрытие информации), а не на интеграцию в тяжеловесные глобальные системы.

Надо отметить, что **STEALTH 2.1** уже получила достаточно широкое распространение. На сегодняшний день создано более 50 частных сетей на всей территории бывшего СССР. При этом пользователями этой системы являются такие известные предприятия, как

- Туркменский АгроБанк (50);
- Челябинский коммерческий земельный банк (30);
- Санкт-Петербургская биржа недвижимости (80);
- МосСтройЭкономБанк (10).

В скобках указано число включенных в сеть ЭВМ. Как видно из примера, **STEALTH** уверенно функционирует во всех регионах, **НЕ ЖАЛУЯСЬ НА КАЧЕСТВО ЛИНИЙ СВЯЗИ**. Кстати, если Московский СтройЭкономБанк развернул не самую крупную сеть, то он установил своеобразный рекорд по ее протяженности, ведь один из подключен-

ных филиалов, осуществляющих обмен данными с Москвой, расположен в Уренгое.

Версия STEALTH 2.1 поддерживает прямой режим обмена информацией, то есть каждый абонент системы может напрямую общаться с любым другим. Но это все хорошо до тех пор, пока трафик в сети не стал слишком напряженным, пока все пользователи применяют однотипные модемы. А в самом деле, что Вы будете делать, если Вам предстоит объединить в единую сеть не только Hayes совместимые, но и уже упоминавшиеся выше модемы Lexand, Мастак, ФизТех и им подобные? И чтобы пользователю было все равно, с каким оборудованием работает его корреспондент?

Вот тут-то и появляется на сцене STEALTH 3.0. Являясь расширением версии 2.1, новый продукт обладает следующими дополнительными возможностями:

- математическое обеспечение узла сети (host ЭВМ) поддерживает одновременную работу до четырех модемов;
- к каждому из каналов Вы можете подключить практически любой из серийно выпускаемых в России "V.23" или Hayes совместимый модем;
- все операции по приему-передаче файлов, в том числе и на host ЭВМ, осуществляются в фоновом режиме.

В результате у Вас впервые появляется возможность связать в единую сеть передачи данных все разнообразные трактовки "стандарта V.23", вне зависимости от исполнения. Далее, Вы можете активизировать на host машине прикладную задачу, например, базу данных и организовать доступ к ней подключающихся в режиме on-line пользователей. И, наконец, самое главное, STEALTH 3.0 позволяет Вам создать свою собственную сеть электронной почты для оперативной передачи коммерческой информации.

Интересно отметить, что и в новой версии значительное внимание уделено обеспечению дружественности интерфейса. Например, ну какое дело отечественному пользователю до сетевого адреса абонента? Тем более, если речь идет о закрытой сети. Он отправляет почту на его имя, пересылая ее на ближайший

Пример 1

```
#            Пример файла управления пакетами
#       Трансляция — SCRIPT.EXE EXAMPLE.SCR
#
# arj:  выбирается для упаковки пакета абоненту с версией 1.5 и выше
# zip:              - для версий 1.4 и ниже
#
arj:                                  # указываем тип используемого архиватора
  для: 355-94-00                      # перечисляем список адресатов пакета,
  для: 266-00-48                      # которых может быть несколько
  инфо:                               # информационная строка — краткое содержание
  файлы:  c:\phone\rsend.*            # перечисляем, какие файлы будут отправлены
          c:\phone\st*.*
  пароль: bankir                      # устанавливаем ключ для шифрования данных
  скорость: средняя                   # и скорость обмена
#
zip:
  для: 374-70-58                      # — с символа # начинается комментарий,
# для: 266-00-48                      #   не обрабатываемый процессором Script
  инфо: Привет вам всем.              # можно передать только краткое сообщение,
  файлы:                              # не передавая никаких файлов
  время: 17:00                        # можно задать время начала передачи
#
unarj:                                # распаковка пакета от абонента с версией 1.5
                                      # и выше
  файлы: albicnfl.arj                 # сюда помещается поступающая почта,
  куда: c:\phone\mail                 # а в этот каталог она будет развернута
#
unarj:
  от: Ком-СТЕЛС                       # можно определить место распаковки почты
  файлы: *.*
  куда: c:\phone\mail\stealth         # от конкретного корреспондента
#
unzip:                                # распаковка пакета от абонента с версией 1.4
  файлы: anbjaggb
  куда: d:\
Стоп:                                 # включение режима Стоп
Прием:                                # включение режима Прием
Посылка:                              # Запуск драйвера в режиме "посылка" с
                                      # сохранением в паузах режима "прием"
```

узел. А узел сам разберется, как ее отправить, чтобы затратить на передачу минимальное время.

Дальнейшее развитие получили и вопросы защиты данных. Фирмой "Ком-СТЕЛС" подписано лицензионное соглашение с известным в кругу специалистов предприятием ЛАН-Крипто на использование системы электронной подписи с открытым ключом. В результате проектом заинтересовались московское ГАИ и московский военкомат. Все это позволяет надяться, что Вооруженные Силы станут крепче, а мафия, наоборот, слабее.

Естественный вопрос, который задают потенциальные покупатели сложных распределенных систем: сопровождение и поддержка пользователей. Надо признать, что к этим проблемам МП "Ком-СТЕЛС" подошло комплексно. Во-первых, Вы можете приобрести либо саму систему, либо с модемами по Вашему выбору, либо, и этот путь представляется самым разумным, Вы можете заказать установку системы "под ключ". В этом случае "Ком-СТЕЛС" проведет анализ линий связи между предполагаемыми абонентами и предложит наиболее экономичное, с точки зрения "эффективность—стоимость", решение. Во-вторых, если в ходе эксплуатации у Вас возникли проблемы, а Вы находитесь на другом конце России, фирма "Ком-СТЕЛС" в состоянии дать Вам техническую консультацию по телефонному каналу! Нет, это будет не телефонный разговор. Уже упоминавшийся модуль дистанционного управления компьютером обеспечивает совместное управление для двух пользователей: сидящего у клавиатуры и подключенного к модему. При этом удаленный пользователь видит перед собой копию экрана управляемого компьютера.

В результате пользователю не надо объяснять, что он видит на экране, а консультанту - что надо нажать. Процесс отладки переходит в совершенно другую плоскость : смотри, как надо! Если возникает необходимость, можно передать в тот или другой конец текстовое сообщение с вопросом или ответом.

Развитие программного продукта поднимает также и вопросы совместимости. Необходимо отметить, что развитие языка сценариев в STEALTH 3.0 позволило сохранить все возможности предыдущей версии, поэтому разработанные для версии 2.1 сценарии могут быть использованы без каких бы то ни было переделок.

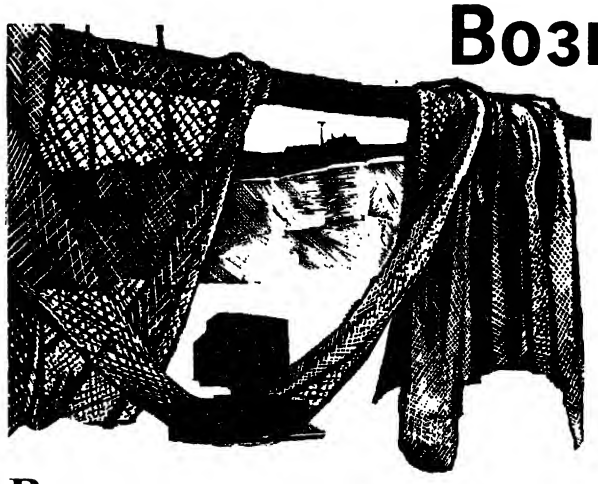
Сложно описать в короткой заметке столь мощное средство, как STEALTH. Но число его пользователей постоянно растет, а результаты работы МП "Ком-СТЕЛС" получают заслуженную оценку. Пакет "Телекоммуникационная сеть STEALTH" неоднократно становился победителем конкурсов программных средств: на специализированных выставках с международным участием "Информационные автоматизированные системы муниципального управления" (июль 1992, Санкт-Петербург), "ИНВЕКОМ-92" (ноябрь 1992, Санкт-Петербург), на Петербургском конгрессе Международного форума информатизации, он имеет почетный диплом Международной Академии Информатизации, сертификаты № 2700/01 и 2700/14.

Ждем Вас на Второй международной выставке "Биржевые и банковские системы" (23-28 февраля 1993 г., 4-й павильон Всероссийского Выставочного Центра (бывш. ВДНХ)).

127635, г. Москва, а/я 12, фирма "Ком-СТЕЛС"
Телефоны: "Ком-СТЕЛС" в Москве
тел./факс (095) 254-32-95; 374-70-58;
"Комэкс" в Санкт-Петербурге
тел. (812) 110-71-49; факс 275-64-84. ■

Возможности СУБД db_VISTA III

И. Кравчинский



В последнее время на рынке программных продуктов для персональных компьютеров типа IBM PC появляется все больше средств (tools) для профессиональных разработчиков, использующих в своей работе стандартные языки программирования, такие как Pascal или C, и ориентирующихся на создание приложений, управляющих доступом к базам данных. Например, CodeBase VI Plus, программно совместимый с dBase III, Paradox Engine 2.0, программно совместимый с Paradox 3.5, и, наконец, db_VISTA III v3.10 фирмы Raima Corp., о которой речь пойдет ниже. Что предпочесть — дело вкуса и склонностей профессионала.

Возможности db_VISTA

Пакет db_VISTA служит инструментарием для программиста, пишущего на языке C, и предназначен для разработки эффективных приложений, создающих СУБД для конечного пользователя. Являясь реализацией сетевой модели данных, СУБД db_VISTA выполнена в соответствии с рекомендациями Комитета по языку описания данных CODASYL. Использование ею в качестве языка программирования C позволяет квалифицированному программисту (ценой, быть может, несколько больших затрат на реализацию приложений) создавать большие и сложные базы данных, критичных по объему и времени выполнения запросов.

Сетевая модель

Сетевая модель — это представление базы данных в виде множества предопределенных записей (объектов), состоящих из полей и объединенных в наборы. Сетевые записи аналогичны реляционным кортежам, они состоят из полей данных, называемых в некоторых реляционных системах атрибутами. Обе модели поддерживают ключевые типы полей, позволяющие идентифицировать запись для быстроты поиска. Кроме записей, сетевая модель реализует любопытный структурный механизм, называемый набор (set).

Использование db_VISTA сетевой модели данных несколько усложняет разработку приложений, но зато обеспечивает программистов, пишущих на C, мощным и эффективным методом организации и обработки сложных информационных структур, позволяющим уже на этапе проектирования базы данных существенно минимизировать избыточность данных.

db_VISTA — реализация сетевой модели

Описание записи в db_VISTA подобно объявлению записи на языке Pascal или структуры C и является шаблоном для группы аналогичных записей. Этот шаблон записи используется затем СУБД для доступа к данным и работы с ними. В дальнейшем изложении мы будем ссылаться на определения записей/наборов как на тип записи/набора, что эквивалентно понятию таблицы в реляционной модели, а на определенную запись/набор в базе как на экземпляр (record).

Набор определяет отношения между типами записей, при которых один из типов записи рассматривается как владелец (owner) по отношению к одному или нескольким типам записей. При этом зависимые типы записей называются элементами или членами (member). db_VISTA реализует наборы только между одним экземпляром владельца и одним или несколькими экземплярами членов.

Такой тип набора называется связью "один-комногим". При этом членами набора могут быть экземляры того же типа, что и хозяин набора.

Наборы реализованы в db_VISTA в виде двухсвязного списка элементов и двух указателей владельца: на начало и конец списка. db_VISTA управляет списками и указателями без участия программиста и делает это очень эффективно. Списки указателей,

с которыми работает db_VISTA, создаются на основании словаря базы данных, создаваемого процессором языка определения данных (ЯОД).

Рисунок 1 иллюстрирует, как db_VISTA реализует связи экземпляров. Квадратики — это записи, а стрелки — указатели. Каждый владелец содержит указатель на начало и конец списка элементов, каждый элемент имеет указатель на предыдущего и на следующего в списке и указатель на его владельца.

Рис. 1. Реализация наборов в db_Vista

Схема (концептуальная модель структуры базы данных) описывается ЯОД спецификациями. При проектировании схемы базы данных следует помнить, что экземпляры наборов образуются привязыванием элементов к владельцу с помощью указателей списка.

Порядок, в котором размещаются элементы, задается описаниями ЯОД.

Процесс добавления нового элемента в набор называется включением записи в набор; аналогично процесс удаления элемента из набора называется исключением.

В db_VISTA в набор могут входить разные типы записей; один тип записи может являться одновременно и владельцем, и элементом различных наборов.

db_VISTA реализует индексацию записей по полям данных, определенным как ключевые поля. B-древовидный индексный файл ключей существенно ускоряет поиск проиндексированных записей. Такое дополнение должно облегчить перестройку мышления программиста при переходе от реляционной модели к сетевой.

Грамматика языка описания данных

Язык определения баз данных синтаксически согласован с C. Грамматика описаний ЯОД приводится ниже. Для описания синтаксиса описаний ЯОД использована нормальная форма Бекуса-Наура (БНФ).

Идентификаторы db_VISTA аналогичны идентификаторам в C, их размер не более 30 символов. Различия между прописными и строчными буквами в них нет, исключая ключевые слова, которые должны писаться строчными буквами. Все идентификаторы должны быть уникальны.

Порядок расположения операторов db_VISTA должен удовлетворять следующим требованиям:

- операторы объявлений key и data файлов должны предшествовать определениям содержащихся в них записей и их полей;
- определения записей должны быть специфицированы до определения содержащих их наборов.

```
1) <ЯОД спецификация>::=
       database <имя базы>    ['['<размер>']']
                              '{'
                              <ЯОД оператор>...
                              '}'

2) <ЯОД оператор>::=
       <спецификация файла> | <спецификация записи>
                            | <спецификация набора>

3) <спецификация файла>::=
       data file <имя файла> contains <имя записи>
```

ЯОД спецификации базы пишутся внутри оператора database. Идентификатор <имя базы> — это имя базы данных, то есть имя создаваемого процессором ЯОД файла словаря базы данных и C include файла. Файл словаря имеет расширение dbd, include файла — h. С параметром <имя базы> работают функция d_open (открыть базу), программа инициализации базы (initdb) и разблокировки (cleardb). <Размер> специфицирует страницы данных (в байтах) файлов базы данных. Это значение ограничивает максимальный размер записи. Его значение по умолчанию 1024.

Оператор ЯОД может быть трех типов. <Спецификация файла> определяет физическую структуру базы данных, <спецификация записи> — тип логической записи, а <спецификация набора> — взаимосвязь логических записей.

Все объявленные записи должны содержаться в data файлах. <Имя файла> — строка, идентифицирующая имя файла, который состоит из записей, специфицированных в списке contains. Один файл может содержать несколько типов записей базы, и каждый тип записи должен содержаться в одном файле.

Специальная системная запись system должна быть указана первой в списке типов записей любого из файлов данных. Она может использоваться как владелец для любого типа записей, как корневая запись сети. При открытии базы она устанавливается текущей записью.

```
4) | key file <имя файла> contains <имя поля> [,<имя поля>].. ';'
```

Все ключевые поля, определенные в ЯОД описаниях записей, должны содержаться в key файле. <Имя файла> — строка, идентифицирующая имя и путь к файлу, содержащему B-древовидные индексы полей, указанные в списке. Как и в случае файлов данных, один файл ключей может содержать несколько типов ключей, определяемых ЯОД, и каждый тип ключевого поля должен содержаться в одном файле.

```
5) <спецификация записи>::=
       record <имя записи>
              '{'
              <спецификация поля>..
              '}'
```

Спецификация записи определяет группу связанных полей данных, обрабатываемых как единое целое. Процессор ЯОД генерирует по параметру <имя записи> два C имени: из строчных букв — имя ярлыка C структуры, аналогичной записи, а из прописных — определение целой константы — номера записи, передаваемого функциям db_VISTA при ра-

боте с записями. Оба эти определения содержатся в h-файле, создаваемом процессором ЯОД.

Допустима спецификация записи, не содержащей полей данных (<спецификация поля>). Такие нулевые записи применяются для организаций связей "многие-ко-многим" через 2 связи "один-ко-многим", в которых нуль-записи являются элементами двух владельцев.

```
6) <спецификация поля>::=
      [[unique] key] <тип> <имя поля> ['['<размер>']']...';'

7) | [[unique] key] struct '{'
                          <спецификация поля>..
                          '}' <имя поля> ['['<размер>']']...';'

8) <тип>::=
      char | int | short | long | float | double | db_adr
```

Синтаксис параметра <спецификация поля> похож на объявления данных в C, большинство допускаемых типов являются типами, непосредственно поддерживаемыми C компилятором. Массивы любых типов задаются указанием спецификации <размер>. Атрибут [unique] key указывает на то, что поле данных будет ключом. Максимальный размер ключевого поля не более 246 байтов.

Поля данных, определенные как unique key, должны все время содержать данные, которые позволяли бы однозначно идентифицировать запись в базе. При попытке создать дубль уникального ключа db_VISTA возвращает статус S_DUPLICATE. Запись при этом не вводится и не модифицируется.

В описании полей не допускаются вложенные структуры. Можно специфицировать только одноразмерный массив структур. Подполя в массиве структур не могут быть определены как ключи.

Тип данных db_adr содержит db_VISTA адреса записей базы, получаемые из таблиц текущего состояния. Он позволяет связывать записи явным образом. Например, можно создать бинарное дерево, явно храня в каждой записи указатели на левое и правое поддерево.

Примечание. Как и в C, в db_VISTA все поля-строки должны заканчиваться нулевым символом. Вот почему поля-строки должны быть на байт больше; они могут содержать в себе '\0', поскольку манипуляция с ними производится через функцию strncpy. В том случае, если данные не удовлетворяют второму требованию, объявляйте их как 2-мерный массив char со второй размерностью.

```
9) <спецификация набора>::=
      set <имя набора>
                '{'
                order <порядок> ';'
                owner <имя записи> ';'
                member <имя записи> [by <имя поля>
                [,<имя поля>]...] ';'
                [member <имя записи> [by <имя поля>
                [,<имя поля>]...] ';']...
                '}'

10) <порядок>::= first | last | ascending | descending | next
```

<Спецификация набора> определяет явственную связь (один-ко-многим) между типами записей. Наборы реализуются привязыванием списка элементов к владельцу, являющемуся корнем (root) куста.

Порядок, в котором записи включаются в этот список, специфицируется пунктом order:

first	— новый элемент включается в начало списка;
last	— новый элемент включается в конец списка;
ascending	— новый элемент включается в порядке увеличения значений полей, указанных в by;
descending	— новый элемент включается в порядке уменьшения значений полей, указанных в by;
next	— новый элемент включается после текущего.

Включение элементов в last, first, next порядке происходит быстрее, так как не требует предварительного сканирования набора, осуществляемого при включении в ascending и descending порядке. В случае больших наборов задержки на сканирование могут стать весьма ощутимыми.

Часть by пункта member нужна только тогда, когда объявлен возрастающий/убывающий порядок следования.

Для наборов, имеющих более одного типа элементов, сортировка производится в соответствии с их типом, длиной и списком by атрибутов. Могут существовать наборы, владелец которых system. В базе существует только одна эта системно определенная запись, являющаяся в этом случае владельцем определенных наборов и служащая корнем сети.

Примечание. При открытии базы данных текущей записью устанавливается system запись, имеющая ссылки на записи типов, определенных в set {...; owner system;..} спецификациях.

Состав db_VISTA и порядок проектирования приложений

db_VISTA состоит из процессора языка определения данных (ЯОД), библиотеки C функций манипулирования данными и нескольких утилит.

Спецификации ЯОД обрабатываются процессором ЯОД db_VISTA, который транслирует их, отслеживая синтаксические ошибки. На выходе процессор ЯОД содает два файла: файл словаря базы данных и include файл определений констант и структуры базы, включаемый в C программу.

Для создания EXE файлов приложений необходимо оттранслировать прикладные программы и связать их с функциями из библиотеки db_VISTA. Библиотечные функции предназначены для работы с базой данных, структура которой определена словарем базы. На рисунке 2 показан процесс создания приложений при работе с db_VISTA.

1) Проектирование базы данных — определение того, как должны быть организованы данные.

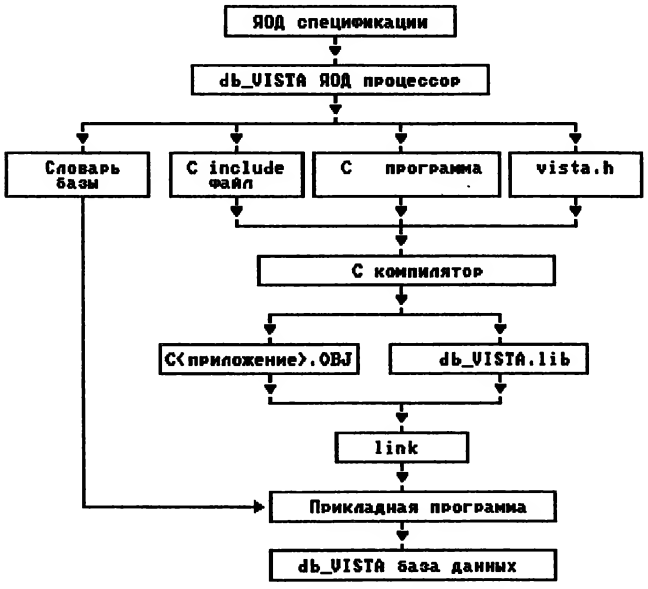


Рис. 2. Схема создания приложений db—Vista III

2) Написание ЯОД спецификаций.

3) Компиляция ЯОД спецификаций DDL (data definition language) процессором:
- исправление ошибок и перекомпилирование;
- экспериментальная проверка (при необходимости) качества организации базы с помощью интерактивного отладчика.

4) Проектирование и программирование С приложений, где манипуляции с базой данных осуществляются вызовами С функций db_VISTA.

5) Компилирование С программ. Необходимо включать в них include файлы vista.h и h файл, сгенерированный ЯОД процессором.

6) Связывание компоновщиком С приложений с библиотекой db_VISTA.

7) Инициализация базы данных (при необходимости) утилитой INITDB.

8) Выполнение приложений.

Предусмотрена уникальная возможность открывать сразу несколько БД и работать с ними одновременно. При этом каждая БД описываетя своей схемой и имеет свой словарь. Для перехода от одной БД к другой не требуется никаких действий, просто в каждой функции обработки данных добавляется еще один параметр — имя БД.

Функции db_VISTA

Все функции db_VISTA можно условно разбить на одиннадцать функциональных групп:
- доступ к БД (открытие, закрытие БД и так далее),
- работа с полями данных,
- работа с записями,
- работа с наборами: соединение и разъединение текущей записи с набором, получение числа членов набора, проверка текущего члена или владельца записи,
- перемещение по набору: поиск текущего владельца, первого, последнего, последующего, предыдущего члена набора,
- поиск по ключевому полю,
- многопользовательское управление БД: определение начала и конца транзакции, блокировка и освобождения файлов записи или набора,
- установка текущей записи, набора или члена,
- доступ к текущей записи, набору или члену,
- получение информации о типе текущей записи, члене или владельце,
- обработка ошибок.

Пример db_VISTA ЯОД определений

Пример 1 спецификаций db_VISTA ЯОД описывает схему сетевой базы данных для типичной структуры организации "отдел/сектор/служащий". Эту схему иллюстрирует рис. 3.

Единственная предопределенная в базе запись system может иметь элементами записи одного или нескольких типов. В нашем случае это записи типа division, а имя набора — div_list. Отделы включаются в базу в порядке возрастания их имен. Отделы состоят из секторов, и эта связь определена набором div_dept. Каждый отдел и сектор имеет своих начальников. Начальники отделов/секторов также являются и служащими, и это отображают наборы div_mgrs и

Пример 1. Описание БД "Корпорация"

```
database corp
                {
                /* БД содержится в одном файле corp.dat */
                data file "corp.dat" contains system, division,
                                department, employee;
                /* все индексы содержатся в файле cop.key */
                key file "corp.key" contains div_name, dep_name,
                                emp_last, emp_id;
record division   { /* описание отдела */
  unique key char div_name[20];
  }
record department { /* описание сектора */
  unique key char dep_name[20];
  }
record employee { /* описание работника */
  unique key char emp_id[6];
  struct
                {
  key char emp_last[20];
  char emp_first[12];
  char emp_middle[12];
            } emp_name;
  char emp_position[12];
  }
set div_list { /* отделы */
  order ascending;
  owner system;
  member division by div_name;
  }
set div_dept { /* секторы отдела */
  order ascending;
  owner division;
  member department by dep_name;
  }
set div_mgrs { /* начальники отделов */
  order first;
  owner division;
  member employee;
  }
set dept_mgrs { /* начальники секторов */
  order first;
  owner department;
  member employee;
  }
set dept_emps { /* служащие сектора */
  order acsending;
  owner department;
  member employee by emp_position, emp_last;
  }
}
```

dept_mgrs. То, что все сотрудники входят в состав секторов, отображает набор dept_emps.

Поиск сотрудников можно осуществлять не только при помощи указанных наборов, но и по уникальному ключу emp_id или ключу emp_last. Отделы и сектора также доступны по ключам.

В результате компиляции corp базы будет создано 2 файла:

1) двоичный файл с именем corp.dbd, содержащий словарь базы;

2) include файл с именем corp.h, используемый С прикладными программами для работы с базой (пример 2).

Этот файл содержит объявления структур каждой записи и макро для имен записей, полей, наборов. Значения этих макро — индекс параметров записей (полей, наборов) в словаре базы данных. Прикладные программы используют эти макро при вызове функций db_VISTA (пример 3).

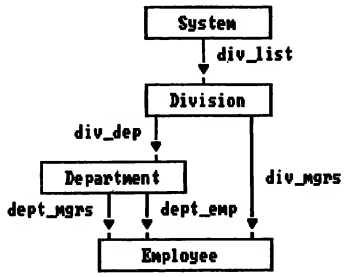


Рис. 3. Схема БД "отдел/сектор/служащий"

Programmer

db_Vista написана на С и переносима на другие типы компьютеров и в другие операционные среды

Достоинства db_VISTA

СУБД, основанная на сетевой модели, — идеал для создателей прикладного программного обеспечения. Возможность определять точные связи между записями предоставляет программисту два преимущества.

1) Возможность проектировать базу с минимальной избыточностью и, как следствие, с большой целостностью.

2) Возможность определять изощренные и эффективные информационные структуры, максимально отвечающие природе информационных процессов.

К основным достоинствам СУБД db_VISTA можно отнести то, что:

- язык определения данных совместим с С;
- данные максимально эффективны, благодаря использованию сетевой модели;
- существует возможность индексации записей по нескольким ключам;
- индексы организованы в виде B-деревьев;
- пользователь полностью контролирует имена файлов и их содержимое;
- при включении записей в набор существует возможность организации автоматической сортировки;
- внутрифайловое пространство базы ведется эффективно, благодаря фиксированной длине записи и повторному распределению освобождаемого пространства;

Пример 2. Содержимое файла corp.h

```
/* Описатели структур, аналогичных записям corp */
struct division {
   char div_name[20];
   };
struct department {
   char dep_name[20];
   };
struct employee  {
   char emp_id[16];
   struct           {
   char emp_last[20];
   char emp_first[12];
   char emp_middle[12];
         } emp_name;
   char emp_position[12];
   };
/* Определение идентификационных номеров полей, записей, наборов */
/* Записей */
#define DIVISION 0
#define DEPARTMENT 1
#define EMPLOYEE 2
#define SYSTEM 3
/* Полей */
#define DIV_NAME 0
#define DEP_NAME 1
#define EMP_ID 2
#define EMP_NAME 3
#define EMP_LAST 4
#define EMP_FIRST 5
#define EMP_MIDDLE 6
#define EMP_POSITION 7
/* Наборов */
#define DIV_LIST 0
#define DIV_DEPT 1
#define DIV_MGRS 2
#define DEPT_MGRS 3
#define DEPT_EMPS 4
```

Пример 3. Программа печати списков служащих корпорации

```
/* Напечатать список управляющих отделами и секторами
в алфавитном порядке. Для каждого отдела напечатать
список секторов и управляющих в этих секторах. */
#include <vista.h>
#include "corp.h"
int main(void)
{
struct employee emp; /* работник */
char dname[32]; /* название отдела/сектора */
/* блокировка всех необходимых наборов */
while (d_setlock(DIV_LIST) = = S_UNAVALL);
while (d_setlock(DIV_DEPT) = = S_UNAVALL);
while (d_setlock(DIV_MGRS) = = S_UNAVALL);
/* печать названий отделов и их управляющих */
for (d_findfm(DIV_LIST); db_status = = S_OKAY; d_findnm(DIV_LIST))
   {
   d_crread(DIV_NAME, dname);   //считать название отдела
   d_setor(DIV_DEPT);           //установить владельца набора div_dept
   d_setor(DIV_MGRS);           //установить владельца набора div_mgrs
   d_findfm(DIV_MGRS);          //найти запись об управляющем отделом
   d_recread(&emp);             //считать запись об управляющем отделом
   printf("Division: %s, Manager: %s %s\n", dname,
         emp.emp_name.emp_first, emp.emp_name.emp_last);
   /* печать сниска секторов и их управляющих */
   for (d_findfm(DIV_DEPT); db_status = = S_OKAY; d_findnm(DIV_DEPT))
      {
      d_crread(DIV_NAME, dname);   //считать название сектора
      d_setor(DEPT_MGRS);          //установить владельца набора dept_mgrs
      d_findfm(DEPT_MGRS);         //найти запись об управляющем
      d_recread(&emp);             //считать запись об управляющем
      printf("Department: %s, Manager: %s %s\n", dname,
            emp.emp_name.emp_first, emp.emp_name.emp_last);
      }
   }
d_freeall(); // отменить все блокировки
}
```

- доступ к диску ускорен за счет виртуализации;
- имеются средства, позволяющие организовать коллективное ведение базы данных (блокировка распределенных файлов и обработка транзакций);
- отсутствует потребность в реорганизации файлов при расширении данных;
- прикладные программы мобильны и невелики по размеру.

db_VISTA написана на C и переносима на другие типы компьютеров. Она с минимальными изменениями транслируется большинством популярных C компиляторов IBM PC и других IBM совместимых компьютеров. Кроме того, она может работать под UNIX в любой системе, имеющей C компилятор, что позволяет создавать легко переносимые приложения.

db_VISTA может работать в среде всевозможных локальных сетей, поддерживающих стандарт NetBIOS.

Если Вы программируете на C и у Вас есть желание добавить в свой арсенал возможность работать с БД, смело выбирайте db_VISTA. Тем самым Вы избавите себя от необходимости изобретать велосипед — разрабатывать свой формат БД и средств для управления ею. Возможность четко определить задачу и средства, необходимые для ее достижения, поможет Вам создавать приложения, имеющие минимальный размер и высокую скорость исполнения, надежное сокрытие данных уменьшит некомпетентное вмешательство пользователя, а переход к объектно-ориентированному программированию (C++) значительно снизит трудозатраты разрабочика, делая их сопоставимыми с трудозатратами программистов, пишущих на dBase или FoxBase.

Далее приведены общие ограничения СУБД и ограничения, накладываемые на размеры. Информация о возможных установках db_VISTA, ограничения, в зависимости от конкретных компьютеров, операционных систем и компиляторов, даны в файле README.

Размер страницы = (по умолчанию) 1024 байта.

Файлы базы db_VISTA разбиты на страницы фиксированной длины, содержащие одну или более записей. Длина может быть изменена при определении базы спецификациями ЯОД.

Минимальный размер записи = размер страницы — 12 байтов.

Запись не может занимать более 1 страницы.

Максимальный размер ключевого поля = 246 байтов.

Максимальное количество записей в файле = 16 777 215.

Максимальное количество файлов = 255.

Количество БД, открытых одновременно, не ограничено ничем, кроме доступной оперативной памяти.

Максимальное количество ЯОД объявлений записей, полей в записи, объявлений наборов, типов элементов наборов, сортировочных полей в наборе практически не ограничено (32.767). Более существенным ограничением является требование на оперативную память под словарь базы, который постоянно хранится в памяти. Размер dbd файла, создаваемого ddl процессором — это, по сути, минимальный размер требуемой db_VISTA дополнительной памяти под открытую базу. ■

Winners

Список победителей конкурса анкет журнала "Монитор"

Читатели, награжденные бесплатной подпиской на журнал "Монитор":

30-летний программист из Санкт-Петербурга Анатолий Евгеньевич Фурса;

32-летний астроном из Санкт-Петербурга В.Е.Абрамов;

24-летний инженер-программист из Львова К.А.Дорошенко;

29-летний программист из Артемовска Луганской обл. В.С.Маценко;

32-летний программист из Гомеля Сергей Юрьевич Кузнецов;

17-летний студент химфака МолдГУ из Кишинева М.М.Пеляху;

35-летний военнослужащий из Мирного Архангельской обл. Марат Рафгатович Сафаров;

40-летний программист из Петропавловска-Камчатского А.В.Тихонов;

24-летний программист из Казани Айрат Сафин;

19-летний студент из пос. Новый Ургал Хабаровского края Леонид Иванович Сизых;

33-летний программист из Приозерска Джезказганской обл. В.О.Полукеев;

14-летний будущий программист из Киева Евгений Середкин;

38-летний океанолог из Севастополя Е.Г.Николаенко;

22-летний инженер из Москвы Г.В.Старынин;

29-летний химик-программист из Москвы А.В.Андрюшин;

26-летний инженер-системотехник из Москвы Ю.Г.Москалев;

39-летний радиоинженер из Москвы Ю.А.Степанов.

Читатели, которые получат по коробке дискет:

29-летний инженер из Челябинска Р.Я.Шакиров;

18-летний студент из Иваново А. Дерягин;

22-летний программист из Самары С.Н.Алехин;

22-летний студент-математик из Петрозаводска Владимир Владимирович Азаров. ■

SYMANTEC

The software side of life

Десять лет истории,
Десять сотен сотрудников,
Десять миллионов пользователей,
И одна идея – Создавать лучшие программы в мире.
Symantec

WHITEWATER RESOURCE TOOLKIT

Вы ищите наиболее современное средство для работы с Windows™? Такое, которое позволит Вам легко управлять и манипулировать ресурсами? Средство, придающее привлекательный и неповторимый облик интерфейсу Вашей программы, И средство, популярное у сотен тысяч разработчиков?

Torga Whitewater Resource Toolkit для Bac! Whitewater Resource Toolkit - экранный редактор для среды Windows™, открывающий новые заманчивые возможности для дизайна интерфейсов программ "под Windows™.

ObjectGraphics

Если же Вы хотели бы найти более легкий способ создавать графические прикладные программы для Microsoft® Windows™, то Вам необходим ObjectGraphics. ObjectGraphics - высокоуровневая, полностью объектно-ориентированная графическая библиотека, придающая дополнительные возможности ObjectWindows, она позволит создать поистине графический интерфейс для Ваших программ. ObjectGraphics - инструмент для пользователей компиляторов Borland® C++, Turbo Pascal ® for Windows™ или Actor®

За подробной информацией о приобретении программных продуктов Symantec обращайтесь к

115409, Москворечье, 31/2
АО "Диалог-МИФИ",
тел 320-3466, факс 324-3055

121165. Москва,
Трио Плюс, Корп.
тел. 281-9890

109044, Крутицкий вал,
3, корп. 2
Merisel-КТ 276-3170, факс 274-0097

123022, Москва
Звенигородское шоссе,9
"Перспективные технологии"
тел. 256-6271

Microsoft - зарегистрированная торговая марка компании Microsoft. Windows - зарегистрированнная торговая марка компании Microsoft. Borland - зарегистрированная торговая марка компании Borland International. Borland C++, Turbo Pascal for Windows, ObjectWindows - зарегистрированные торговые марки компании Borland International. Actor, ObjectGraphics, Whitewater Resource Toolkit - зарегистрированные торговые марки компании Symantec Corporation.

Мини–СУБД в Вашей программе

А.Сунгатуллин

Почти каждый серьезный разработчик программного обеспечения сталкивался с необходимостью использования тех или иных средств работы с базами данных (БД) в своих программах. Существует весьма обширный класс задач, требующих мощных инструментальных средств для организации хранения и обработки больших массивов данных, например, программные комплексы автоматизированного управления производством или банковские системы.

Для этих целей разработаны специальные программные продукты, называемые системами управления базами данных (СУБД). Эти системы предоставляют мощные средства для интерактивной работы с БД и для администратора БД.

Как правило, СУБД включают в себя специализированные языки программирования приложений, работающих с БД. Среди таких языков можно упомянуть SQL, который стал практически стандартом для работы с данными в сетях ЭВМ. В среде MS-DOS наибольшее распространение получил язык XBASE. Примерами XBASE совместимых СУБД могут служить широко известные и рекламируемые в нашей стране пакеты Clipper 5.01, dBASE IV 1.5, FoxPro 2.0. Кроме того, уверенно набирает силу Paradox, поддерживающий свой собственный формат данных и язык разработки приложений PAL.

Необходимо заметить, что описанный класс СУБД больше подходит для интерактивной работы с базами данных и для написания несложных программ непрограммирующим пользователем. Языковые средства этих программ сильно ограничены и практически позволяют только решить задачу управления данными и организовать простой интерфейс с пользователем. Этого явно недостаточно для разработки на современном уровне крупной системы с мощными средствами представления информации и связи. Кроме того, сетевые возможности этих программ очень слабо развиты, да и вопрос об эффективности реляционной модели данных в сетевых приложениях на сегодняшний день не решен.

Другой класс инструментальных средств для работы с БД - это более сложные и эффективные продукты, представляющие собой либо СУБД,обеспечивающие программные интерфейсы с универсальными языками программирования, как правило, с языком С, либо библиотеки для работы с БД для тех же языков высокого уровня. Примерами продуктов этого класса могут служить СУБД HiTech, db_Vista, MDBS, библиотеки CodeBase, C-tree, PforCe и Paradox Engine.

Большинство этих средств позволяет создавать эффективные однопользовательские и сетевые приложения, а также обеспечивает высокую степень переносимости программ в различные операционные среды на компьютеры многих типов. Практически для любых, даже несложных задач обработки данных применение продуктов данного класса более предпочтительно. Современные системы разработки программ на языках С и Паскаль, в сочетании с сотнями коммерческих библиотек самого разного назначения, дают ни с чем не сравнимые по гибкости и мощи средства для организации интерфейса с пользователем, работы с разнообразными периферийными устройствами, доступа к аппаратуре и функциям операционной системы, чего нельзя сказать о современных специализированных СУБД. Что касается удобства манипулирования базами данных в стандартных языках высокого уровня, то это зависит лишь от того, насколько хорошо программист владеет тем или иным языком программирования.

Программы, предназначенные для решения задач математического моделирования, инженерных расчетов, автоматического проектирования или обработки графических данных, как правило, должны хранить, обновлять, сортировать или выводить некоторые данные, например, поддерживать базу данных результатов расчета или стандартного набора элементов для проектирования. Такие программы обычно содержат большое количество вычислительных операций, часто в реальном масштабе времени, интенсивно используют память и аппаратные особенности компьютера, а иногда к тому же работают в защищенном режиме процессора. Все это делает невозможным применение специализированных СУБД для реализации таких программ. Здесь обычно применяются

библиотеки функций универсальных языков программирования, предназначенные для работы с БД.

Перечисленные выше программные продукты для работы с БД представляют собой специализированные и достаточно сложные в изучении инструменты, применение которых в Вашей разработке требует серьезного обоснования, если, конечно, Вы задумываетесь об эффективности Вашей программы. К тому же эти средства являются коммерческими продуктами, причем стоимость их исчисляется сотнями, а иногда и тысячами долларов, и не все из них можно купить за рубли.

Если же Вы разрабатываете резидентный справочник или словарь, программу-секретарь с резидентным дневником или коммуникационный пакет с телефонной книгой, то часто не удается использовать даже специализированные библиотеки для манипулирования данными, так как при этом, по опыту автора, исполняемый файл Вашей программы увеличится на 40-60 килобайтов, что может быть недопустимым.

В данной статье представлена библиотека функций для работы с БД (исходные тексты на языке С и демонстрационная программа, иллюстрирующая ее применение).

Предложенная библиотека разработана автором и опубликована с целью сделать ее исходный текст общественной собственностью, доступной для широкого круга программистов-практиков. Библиотека является завершенным и готовым к применению программным средством. Вы можете свободно использовать эти исходные тексты в своих разработках, вносить в них изменения с целью развития и улучшения.

В состав библиотеки входят два файла: MINIBASE.C - файл с исходными текстами функций на языке С (листинг 1), MINIBASE.H (листинг 2) — файл с определениями констант и прототипами пользовательских функций библиотеки, который должен быть включен директивой #include в исходный текст Вашей программы для использования функций MINIBASE. Данный файл также содержит краткие описания пользовательских функций, облегчающие их применение. Кроме того, в статье приведен текст демонстрационной программы PHONE (листинг 3), использующей библиотеку MINIBASE для организации телефонного справочника. Вы можете получить исполняемый файл программы при помощи вашего компилятора С. Ниже приведены образцы командной строки для компиляции программы наиболее известными компиляторами.

```
Zortech C++ v.2.1,3.0:   ztc -msi phone.c minibase.c

Microsoft C v.5.1,6.0:   cl phone.c minibase.c

Borland C++ v.2.0,3.0:   bcc -w- -f- phone.c minibase.c
```

Библиотека MINIBASE достаточно проста и имеет серьезные ограничения на размеры и структуру поддерживаемых БД, но ее возможностей может оказаться достаточно для решения Вашей конкретной задачи по управлению данными с минимальными накладными расходами. Приведенное ниже обсуждение достоинств и недостатков MINIBASE позволит Вам определиться в отношении целесообразности использования библиотеки в Ваших разработках.

Библиотека позволяет создавать индексированные файлы БД заданной структуры, открывать и закрывать одновременно несколько БД в разных рабочих областях (dbselect,dbuse,dbclose), перемещать указатель файла на любую запись последовательно или по номеру (dbgoto,dbnext,dbprev), читать, обновлять, добавлять и удалять записи БД (dbread, dbwrite, dbappend, dbdelete), осуществлять быстрый поиск записи в БД по ключу (dbseek), а также получать различную информацию о состоянии БД (dbreccount, dbrecno, dbeof). БД может быть проиндексирована по числовому значению типа long, по символьной строке произвольной длины или строке, содержащей дату в формате дд/мм/гг, причем разделители даты могут быть любыми. При этом библиотека имеет ряд отличительных достоинств, которые могут оказаться существенными.

1) Данное средство является самым компактным инструментом для работы с БД из известных автору. Использование MINIBASE в Вашей разработке приведет к увеличению исполняемого файла всего на 6 килобайтов.

2) Исходный текст библиотеки абсолютно удовлетворяет требованиям стандарта ANSI C и содержит вызовы функций только стандартной библиотеки, что позволяет использовать его без модификаций с любым компилятором С (Zortech C/C++, Microsoft C, QuickC, Turbo C/C++, Borland C/C++ и так далее) в любой модели памяти, а также на различных компьютерах практически с любой операционной системой (MS-DOS, OS/2, XENIX, UNIX, VMS...).

3) Использование MINIBASE для работы с БД не приведет к тому, что Вашу программу уличат в "порочных связях" с какой-нибудь из коммерчески поставляемых библиотек, добытых нелегальным копированием.

4) Вы всегда можете модифицировать библиотеку и обогатить ее новыми функциями, имея полный исходный текст, который отличается простотой и небольшим объемом.

Ниже приведены рабочие характеристики библиотеки; достаточны ли они, зависит от конкретной задачи. Максимальное количество одновременно открытых БД определяется числом доступных файловых указателей и в общем случае не может быть больше 14; максимальный размер записи не должен превышать 32 килобайта; каждая БД может иметь только один ключ оговоренных длины и типа, причем за ключ автоматически принимаются первые N байтов записи, N определяется типом и размером ключа; количество записей в БД теоретически ограничено числом 32000, но практически оно сильно ограничено доступной оперативной памятью, размером ключа и количеством одновременно используемых БД, так

что использование библиотеки целесообразно при номинальных размерах баз данных не более 2000 записей; кроме того, формат БД является собственным для MINIBASE, хотя и чрезвычайно простым.

Автор надеется, что его коллеги-программисты, для которых и была написана данная статья, получат практическую пользу от использования описанной технологии в своих разработках. ■

Список рекомендуемой литературы

1. М.Р. Когаловский. Технология баз данных на персональных ЭВМ. - М.: Финансы и статистика, 1992.
2. Инструментальные средства языка Си для разработки баз данных // Журнал доктора Добба. - 1991. - N 4.
3. СУБД для локальных вычислительных сетей // Журнал доктора Добба. - 1991. - N 2.
4. СУБД HiTech - шаг к реальному масштабу времени // Мир ПК. - 1991. - N 8.

КОМПЛЕКС ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ

QUICKTOOLS FOR PARADOX версия 1.0

ТОО 'ИНТЕГРИРОВАННЫЕ КОМПЬЮТЕРНЫЕ СИСТЕМЫ'

Если Вы разрабатываете прикладные системы на базе СУБД Paradox 3.5, то QuickTools 1.0 for Paradox - инструмент для Вас ! То, на что у Вас раньше уходили недели, теперь Вы сможете сделать за минуты и часы.

По своим возможностям QuickTools не уступает, а кое в чем и превосходит Paradox 4.0 WorkShop. При этом в отличии от Paradox 4.0 WorkShop QuickTools под управлением Paradox 3.5 с успехом работает на PC/AT с 1 Мб RAM.

QuickTools - это комплекс инструментальных средств, которые позволяют автоматизировать разработку интерфейса прикладных систем.

Разработчики приложений могут использовать QuickTools для моделирования сложных систем, уменьшая объем кодирования на языке PAL и снижая тем самым трудоемкость их разработки.

Используя QuickTools1.0 Вы можете.

- Создавать горизонтальные, вертикальные и матричные меню.
- Связывать с элементами меню действия, активизируемые при их выборе.
- Формировать заставки (TitleScreens).
- Создавать экраны Help и связывать их с элементами меню.
- Получать программы на языке PAL, осуществляющие вывод заставок, меню, подсказок, и активизировать действия, связанные с выбором в меню.
- Использовать библиотеку типовых процедур на языке PAL.

PARADOX, PAL, WorkShop - торговые марки или зарегистрированные торговые марки Borland International, Inc.

За справками обращаться : тел. 238-99-33; факс 238-32-78; адрес 117419, Москва а/я 765.

Монитор — 1993 Подписной купон Монитор — 1993

Адрес _____

Ф.И.О. или название организации _____

МОНИТОР

Стоимость подписки на 1 год (10 номеров) — 1200 рублей. Перечислите сумму оплаты на счет НТО "Софт-Москва": р/с 467002 в КБ "Банк развития XXI век", к/с 161816 в РКЦ ГУ ЦБ РФ по г. Москве, МФО 201791

Вышлите подписной купон и квитанцию об оплате по адресу: 117419, Москва, а/я 765.

Листинг 1

```
/*****
*
* MINIBASE.C
*
* Тексты функций библиотеки MINIBASE
* Альберт Сунгатуллин (C) 1992 Москва
*
*****/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "minibase.h"

#ifndef __ZTC__
#include <malloc.h>
#include <memory.h>
#endif

static int  indcmp( void *e1, void *e2 );
static int  indcmpdate( char *date1, char *date2 );
static short    indupdrec();
static void     indsort();
static short    recuse( char *fname, short recsiz, short keysiz );
static void     recclose();
static short    reccount();
static short    recno();
static short    recerror();
static short    receof();
static short    recgoto( short number );
static short    recprev();
static short    recnext();
static short    recread( void *dest );
static short    recwrite( void *src );
static short    recappend( void *src );
static short    recreadkey( void *src );

static short    workarea = 0, error = 0;
static struct   dbinfo db[FILES+1];
static struct   indinfo ind[FILES];

/***
*
* Внешние (пользовательские) функции библиотеки MINIBASE
*
***/

short dbselect( short area )
{
    error = 0;
    if( area < 1 || area > FILES+1 ) error = 1;
    else  workarea = area-1;
    return error;
}

short dbuse( char *fname, short recsiz, short keysiz, short ktype )
{
    FILE *indfile;
    char *ptr;
    short cc,count,*rp;

    error = ind[workarea].deleted = ind[workarea].indno = 0;

    if( ktype < 1 || ktype > 3 )   ind[workarea].keytype = KLINE;
    else  ind[workarea].keytype = ktype;

    if( recuse( fname, recsiz, keysiz )! = 0 )
    {
        error = 1;
        return error;
    }
    if( reccount() > MAXREC )
    {
        recclose();
        error = 1;
        return error;
    }
    ind[workarea].indcount = count = reccount();
    strcpy( ind[workarea].name, db[workarea].name );

    for( ptr = ind[workarea].name+strlen(ind[workarea].name)-1;
            ptr > ind[workarea].name && *ptr! = ':' && *ptr! = 0x5C;
            ptr--
        )
    if( *ptr = = '.' ) { *ptr = 0; break; }
    strcat( ind[workarea].name, INDEXT );
    if( count = = 0 )       remove( ind[workarea].name );

    ind[workarea].mb = malloc(
(db[workarea].keysize+sizeof(short))*MAXREC );
    if( ind[workarea].mb = = NULL )
    {
        recclose();
        error = 1;
        return error;
    }
    for( cc = 0; cc < count; cc++ )
    {
        ind[workarea].tb[cc] = (char *)ind[workarea].mb+
        cc*(db[workarea].keysize+sizeof(short));
    }
    indfile = fopen( ind[workarea].name, "rb" );
    if( indfile! = NULL )
    {
        fread( ind[workarea].mb, db[workarea].keysize+sizeof(short),
                    count, indfile );
        fclose( indfile );
    }
    else
    {
        for( cc = 0; cc < count; cc++ )
        {
            rp = (short *)((char *)ind[workarea].tb[cc]+db[workarea].keysize);
            recgoto( cc+1 );
            recreadkey( ind[workarea].tb[cc] );
            *rp = recno();
        }
    }
    indsort();
    return error;
}

void dbclose()
{
    FILE *indfile;
    short wa,count,failure;
    void *bf;

    indsort();
    count = dbreccount();

    if( count = = 0 )
    {
        free( ind[workarea].mb );
        recclose();
        remove( db[workarea].name );
        remove( ind[workarea].name );
        return;
    }
    if( !ind[workarea].deleted )
    {
        indfile = fopen( ind[workarea].name, "wb" );
        if( indfile! = NULL )
        {
            dbgoto( 1 );
            while( !dbeof() )
            {
                fwrite( ind[workarea].tb[dbrecno()-1],
                                db[workarea].keysize+sizeof(short),
                                1, indfile );
                dbnext();
            }
            fclose( indfile );
        }
        free( ind[workarea].mb );
        recclose();
        return;
    }
    remove( INDTMP );
    remove( ind[workarea].name );

    bf = malloc( db[workarea].recsize );
    if( bf = = NULL )
    {
        free( ind[workarea].mb );
        recclose();
        return;
    }
    wa = workarea;
    dbselect( FILES+1 );
    if( recuse( INDTMP, db[wa].recsize, db[wa].keysize )! = 0 )
    {
```

```
            free( ind[wa].mb );
            free( bf );
            dbselect( wa+1 );
            recclose();
            return;
        }
        dbselect( wa+1 );
        dbgoto( 1 );

        while( !dbeof() )
        {
            failure = dbread( bf );
            if( failure )      break;
            dbselect( FILES+1 );
            failure = recappend( bf );
            if( failure )      break;
            dbselect( wa+1 );
            dbnext();
        }
        dbselect( FILES+1 );
        free( bf );
        recclose();
        free( ind[wa].mb );
        dbselect( wa+1 );
        recclose();

        if( !failure)
        {
            remove( db[wa].name );
            rename( INDTMP, db[wa].name );
        }
        return;
}

short dbreccount()  { return ind[workarea].indcount; }

short dbrecno()     { return ind[workarea].indno; }

short dbeof()
{
        if( ind[workarea].indno = = 0 )      return 1;
        return 0;
}

short dbgoto( short number )
{
        error = 0;
        if( number < 1 || number > dbreccount() )
        {
            error = 1;
            ind[workarea].indno = 0;
        }
        else  ind[workarea].indno = number;

        indupdrec();
        return error;
}

short dbprev()
{
        error = 0;
        if( dbrecno() = = 1 )
        {
            error = 1;
            return error;
        }
        if( dbeof() )    return dbgoto( dbreccount() );
        else
        {
            --ind[workarea].indno;
            indupdrec();
        }
        return error;
}

short dbnext()
{
        error = 0;
        if( dbeof() )
        {
            error = 1;
            return error;
        }
        ++ind[workarea].indno;
        if( ind[workarea].tb[dbrecno()-1] = = NULL )      dbgoto( 0 );
        indupdrec();
        return error;
}

short dbread( void *dest )
{
        indupdrec();
        return recread( dest );
}

short dbwrite( void *src )
{
        short ci,count,*rp;

        error = 0;
        indupdrec();

        ci = dbreccount();
        count = reccount();

        if( dbeof() )
        {
            if( count > = MAXREC )
            {
                error = 1;
                return error;
            }
            ind[workarea].tb[ci] = (char *)ind[workarea].mb+
            count*(db[workarea].keysize+sizeof(short));
            rp = (short *)((char *)ind[workarea].tb[ci]+db[workarea].keysize);
            recappend( src );
            if( error ) return error;
            ind[workarea].indcount++;

            recreadkey( ind[workarea].tb[ci] );
            *rp = recno();
            dbgoto( ci+1 );
            return 0;
        }
        else
        {
            recwrite( src );
            if( recerror() ) return error;
            recreadkey( ind[workarea].tb[dbrecno()-1] );
        }
        return error;
}

short dbappend( void *src )
{
        dbgoto( 0 );
        return dbwrite( src );
}

short dbdelete()
{
        short num;

        error = 0;
        if( dbeof() )
        {
            error = 1;
            return error;
        }
        ind[workarea].deleted = 1;
        num = dbrecno();
        ind[workarea].tb[num-1] = NULL;
        indsort();
        ind[workarea].indconnt--;
        dbgoto( num );
        return 0;
}

void dbseek( void *key )
{
        void *result, **kptr;
        short place, count;

        indsort();

        count = dbreccount();
        if( count = = 0 )      return;

        kptr = &key;
        result = bsearch( kptr, (void *)ind[workarea].tb, count,
                                        sizeof(void *), (int (*)())indcmp );

        if( result = = NULL )
```

Листинг 1 (продолжение)

```
        {
            dbgoto( 0 );
            return;
        }
        place = ((char *)result-(char *)ind[workarea].tb)/sizeof(void *)+1;
        dbgoto( place );

        while( dbrecno() > 1 )
        {
            dbprev();
            if( indcmp( kptr, &ind[workarea].tb[dbrecno()-1] )!= 0 )
            {
                dbnext();
                break;
            }
        }
        return;
}

/***
*
*  Внутренние функции библиотеки MINIBASE
*
***/

void indsort()
{
        qsort( ind[workarea].tb, reccount(), sizeof(void *), (int (*)())indcmp );
        dbgoto( 1 );
        error = 0;
        return;
}

int indcmp( void *e1, void *e2 )
{
        void **pt1, **pt2;
        long *pu1, *pu2;
        char *ps1, *ps2;

        pt1 = e1;
        pt2 = e2;

        if( *pt1 == NULL && *pt2 == NULL ) return 0;
        if( *pt1 == NULL ) return 1;
        if( *pt2 == NULL ) return -1;

        switch( ind[workarea].keytype )
        {
            case KLONG:
                pu1 = (long *)*pt1;
                pu2 = (long *)*pt2;
                if( *pu1 == *pu2 ) return 0;
                if( *pu1 > *pu2 )  return 1;
                return -1;
            case KLINE:
                ps1 = (char *)*pt1;
                ps2 = (char *)*pt2;
                return  memcmp( ps1, ps2, db[workarea].keysize );
            case KDATE:
                ps1 = (char *)*pt1;
                ps2 = (char *)*pt2;
                return  indcmpdate( ps1, ps2 );
            default:
                break;
        }
        return 0;
}

int indcmpdate( char *date1, char *date2 )
{
        char bf[3];
        long dval1, dval2, year, month, day;

        bf[0] = date1[0];  bf[1] = date1[1];  bf[2] = 0;
        day = atoi( bf );

        bf[0] = date1[3];  bf[1] = date1[4];  bf[2] = 0;
        month = atoi( bf );
        month = month * 40;

        bf[0] = date1[6];  bf[1] = date1[7];  bf[2] = 0;
        year = atoi( bf );
        year = year * 500;

        dval1 = day + month + year;

        bf[0] = date2[0];  bf[1] = date2[1];  bf[2] = 0;
        day = atoi( bf );

        bf[0] = date2[3];  bf[1] = date2[4];  bf[2] = 0;
        month = atoi( bf );
        month = month * 40;

        bf[0] = date2[6];  bf[1] = date2[7];  bf[2] = 0;
        year = atoi( bf );
        year = year * 500;

        dval2 = day + month + year;

        if( dval1 > dval2 )  return 1;
        if( dval1 < dval2 )  return -1;
        return 0;
}

short indupdrec()
{
        short *rp;
        if( dbrecno() == 0 )return recgoto( 0 );
        rp = (short *)((char *)ind[workarea].tb[dbrecno()-
1]+db[workarea].keysize);
        return recgoto( *rp );
}

short recuse( char *fname, short recsiz, short keysiz )
{
        error = 0;
        strcpy( db[workarea].name, fname );

        db[workarea].recsize = recsiz;
        db[workarea].keysize = keysiz;
        db[workarea].recno = 1;

        db[workarea].rec = fopen( fname, "r+b" );
        if( db[workarea].rec == NULL )
        {
            db[workarea].rec = fopen( fname, "w+b" );
            db[workarea].recno = 0;
        }
        if( db[workarea].rec == NULL )    error = 1;
        return error;
}

void recclose()
{
        short empty = 0;
        if( reccount() == 0 ) ++empty;
        fclose( db[workarea].rec );
        if( empty )       remove( db[workarea].name );
        return;
}

short reccount()
{
        long now,end;
        short count;

        now = ftell( db[workarea].rec );
        fseek( db[workarea].rec, 0L, SEEK_END );
        end = ftell( db[workarea].rec );
        end = end/db[workarea].recsize;
        count = end;
        fseek( db[workarea].rec, now, SEEK_SET );
        return count;
}

short recno()   { return db[workarea].recno; }

short recerror()      { return error; }

short receof()
{
        if( db[workarea].recno == 0 )return 1;
        return 0;
}

short recgoto( short num )
{
        short number;
        long where;

        number = num;
        error = 0;
```

```
    if( number < 1 || number > reccount() )
    {
        error = 1;
        fseek( db[workarea].rec, 0L, SEEK_END );
        db[workarea].recno = 0;
    }
    else
    {
        where = number-1;
        where = where*db[workarea].recsize;
        fseek( db[workarea].rec, where, SEEK_SET );
        db[workarea].recno = number;
    }
    return error;
}

short recprev()
{
    error = 0;
    if( recno() = = 1 || reccount() = = 0 )
    {
        error = 1;
        return error;
    }
    if( receof() )   recgoto( reccount() );
    else  recgoto( recno()-1 );
    return error;
}

short recnext()
{
    error = 0;
    if( receof() )
    {
        error = 1;
        return error;
    }
    return recgoto( recno()+1 );
}

short recread( void *dest )
{
    short count;
    long back;

    error = 0;
    if( receof() )
    {
        error = 1;
        return error;
    }
    count = fread( dest, db[workarea].recsize, 1, db[workarea].rec );
    back = -db[workarea].recsize;
    fseek( db[workarea].rec, back, SEEK_CUR );
    if( count! = 1 ) error = 1;
    return error;
}

short recwrite( void *src )
{
    short count;

    error = 0;
    count = fwrite( src, db[workarea].recsize, 1, db[workarea].rec );
    if( receof() )   recgoto( reccount() );
    else  recgoto( recno() );
    if( count! = 1 ) error = 1;
    return error;
}

short recappend( void *src )
{
    recgoto( 0 );
    return recwrite( src );
}

short recreadkey( void *src )
{
    short recsiz;
    recsiz = db[workarea].recsize;
    db[workarea].recsize = db[workarea].keysize;
    recread( src );
    db[workarea].recsize = recsiz;
    return error;
}
```

■

▼ Листинг 2

```
/*****
*
* MINIBASE.H
*
* Заголовочный файл библиотеки MINIBASE
* Альберт Сунгатуллин (C) 1992 Москва
*
*****/


#ifndef   MINIBASE_H
#define   MINIBASE_H

#define   FILES     1          /* количество одновременно открытых БД */
#define   MAXREC 1000          /* максимальное количество записей в БД */

#define   INDTMP  "MINIBASE.TMP"  /* файл промежуточного вывода */
#define   INDEXT  ".IND"          /* расширение для индексных файлов */

#define   KLONG   1          /* константы, определяющие тип ключа */
#define   KLINE   2
#define   KDATE   3

struct dbinfo
{
    char name[130];
    FILE *rec;
    short recsize;
    short keysize;
    short recno;
};

struct indinfo
{
    char name[130];
    short keytype;
    short indno;
    short indcount;
    short deleted;
    void *mb;
    void *tb[MAXREC];
};

/***
*
* Пользовательские функции библиотеки MINIBASE
*
***/

/*
* DBSELECT()
* устанавливает активную рабочую область <area> в диапазоне от 1
* до FILES; возвращает 0, если нет ошибки; все остальные функции
* оперируют с БД в установленной области.
*/
short dbselect( short area );

/*
* DBUSE()
* открывает БД с именем <fname>, размером записи <recsiz>, длиной
* ключа в байтах <keysiz> в активной рабочей области; <ktype>
* определяет тип ключа (KLONG, KLINE или KDATE); открывается также
* индексный файл с расширением INDEXT, а если его нет, пере-
* индексируется БД; если файл БД с указанным именем не найден,
* создает пустой файл БД; возвращает 0, если нет ошибки.
*/
short dbuse( char *fname, short recsiz, short keysiz, short ktype );

/*
* DBCLOSE()
* закрывает БД в активной рабочей области.
*/
void dbclose();

/*
* DBRECCOUNT()
* возвращает число записей БД в активной рабочей области.
*/
short dbreccount();

/*
* DBRECNO()
* возвращает номер текущей логической записи БД в активной
* рабочей области; возвратит 0, если указатель стоит на конце
* файла.
*/
```

▶

Листинг 2 (продолжение)

```
short dbrecno();

/*
* DBEOF()
* возвращает ненулевое значение, если указатель стоит на конце
* файла.
*/
short dbeof();

/*
* DBGOTO()
* перемещает указатель на запись с номером <number>;
* если <number> равен 0 или больше существующего,
* перемещает указатель на конец файла и возвращает
* ненулевое значение.
*/
short dbgoto( short number );

/*
* DBPREV()
* перемещает указатель на предыдущую логическую запись;
* возвращает 0, если действие успешно; если указатель
* стоял на первой записи, или файл БД пуст, возвращается
* ненулевое значение.
*/
short dbprev();

/*
* DBNEXT()
* перемещает указатель на следующую логическую запись;
* если указатель стоял на последней записи, он перемещается
* на конец файла, а если он стоял на конце файла, то
* возвращается ненулевое значение.
*/
short dbnext();

/*
* DBREAD()
* читает текущую запись БД и записывает ее по адресу <dest>;
* возвращает ненулевое значение, если указатель стоял на
* конце файла или произошла ошибка при чтении.
*/
short dbread( void *dest );

/*
* DBWRITE()
* записывает данные из буфера <src> в текущую запись; если
* указатель стоял на конце файла - добавляет новую запись;
* возвращает 0, если не было ошибки при записи.
*/
short dbwrite( void *src );

/*
* DBAPPEND()
* работает так же, как и DBWRITE(), но добавляет запись всегда
* в конец файла, независимо от положения указателя.
*/
short dbappend( void *src );

/*
* DBDELETE()
* удаляет из БД текущую запись; если указатель стоял на
* конце файла - возвращает ненулевое значение.
*/
short dbdelete();

/*
* DBSEEK()
* осуществляет быстрый поиск записи в БД, значение ключа
* которой полностью совпадает со значением по адресу <key>
* и перемещает указатель на найденную запись; если такая
* запись не найдена, перемещает указатель записи на конец
* файла; если существует несколько записей с заданным ключом,
* будет найдена самая верхняя запись.
*/
void dbseek( void *key );

#endif
```

Листинг 3

```
/*****
*
* PHONE.C
*
* Программа, демонстрирующая использование MINIBASE
* Альберт Сунгатуллин (С) 1992 Москва
*
*****/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "minibase.h"

main( int argc, char *argv[] )
{
    char context[80];
    short err;
    struct
    {
        char name[40];
        char number[30];
    } phone;
    dbselect( 1 );
    err = dbuse( "PHONE.REC", sizeof(phone), sizeof(phone.name), KLINE );
    if( err )
    {
        printf("\r\nОшибка при открытии (создании) базы данных !\r\n");
        exit( 1 );
    }
    strupr( argv[1] );
    memset( &phone, 0, sizeof(phone) );
    memset( context, 0, sizeof(context) );
    printf("\r\n<Телефонная книга>  <Абонентов-
%d>\r\n\n",dbreccount());

    switch( argv[1][1] )
    {
        case 'A':
            printf("Абонент: ");
            gets( phone.name );

            if( strlen( phone.name ) == 0 )
            {
                printf("Абонент не записан !\r\n");
                break;
            }

            printf("Телефон: ");
            gets( phone.number );
            dbappend( &phone );
            break;
        case 'D':
            printf("Абонент: ");
            gets( phone.name );
            dbseek( phone.name );
            if( dbeof() )    printf("Абонент не найден !\r\n");
            else  dbdelete();
            break;
        case 'F':
            printf("Контекст для поиска: ");
            gets( context );
            if( strlen( context ) == 0 ) break;
            printf("\r\n");
        case 'L':
            dbgoto( 1 );
            while( !dbeof() )
            {
                dbread( &phone );
                if( argv[1][1] == 'F' ||
                        strstr(phone.name,context) != NULL ||
                        strstr(phone.number,context) != NULL
                ) printf("%-40s %-30s\r\n",phone.name,phone.number);
                dbnext();
            }
            break;
        default:
            printf("Использование: PHONE /Ключ\r\n");
            printf("/A.....добавить абонента\r\n");
            printf("/D.....удалить абонента\r\n");
            printf("/L.....список абонентов\r\n");
            printf("/F.....найти по контексту\r\n");
        break;
    }
    dbclose();
    exit( 0 );
}
```

"НАТЕКС" — наступление продолжается!

Мы уже писали о московской фирме "Натекс", освоившей серийное производство Hayes совместимых модемов (Монитор, 1992, № 3). Первая попытка оказалась достаточно успешной, модем UNICOM HAYES 2400 получил сертификат Министерства Связи России и ... возникли проблемы с производством. Собственные мощности предприятия работают на полную мощность, модем поставлен на конвейер, но (!) СПРОС ЗНАЧИТЕЛЬНО ПРЕВЫШАЕТ ПРЕДЛОЖЕНИЕ! Это значит, что покупатель модемов вынужден становиться в очередь, или приобретать импортный модем с теми же характеристиками, но за дополнительные деньги. В результате одна из первоочередных задач, стоящих перед фирмой "НАТЕКС", — поиск предприятия, готового освоить серийный выпуск модемов фирмы с объемом производства до 1000 штук в месяц.

Но вернемся к проблемам пользователей ЭВМ и программистов-практиков. Не останавливаясь на достигнутом, "НАТЕКС" менее чем за год разработал и выпустил на рынок новую модель — внешний модем UNICOM HM 2400EV. Эта модель не только поддерживает протоколы CCITT V.22bis, V.22, V21, Bell 103,212A, но и реализует аппаратную коррекцию ошибок (MNP 4, CCITT V.42) и аппаратное сжатие данных (MNP5, CCITT V.42bis). Новое изделие реализовано на сигнальном процессоре фирмы INTEL 89C024FT. Имеется возможность регулировки чувствительности и уровня выходного сигнала, что значительно расширяет возможности по адаптации изделия к конкретным телефонным линиям. Стоимость изделия составляет $120 при розничной продаже и $95 при поставке партиями от 100 штук (оплата производится в рублях по текущему курсу).

Следующей приятной новостью для пользователей может служить увеличение гарантийного срока службы до трех лет для уже отлаженной серийной модели UNICOM HAYES 2400 и одногодичный гарантийный срок для новой модели UNICOM HM 2400EV. При этом российская сеть сервиса фирмы "НАТЕКС" обеспечивает поддержку и ремонт всех выпускаемых моделей.

Если прошлое изделие (UNICOM HAYES 2400) было реализовано на микросхемах фирмы Rockwell International Corp., то новая модель собрана на базе ИМС фирмы Intel. Это связано с тем, что в настоящее время фирма Rockwell International Corp. при участии фирмы "НАТЕКС" проводит исследование характеристик отечественных телефонных каналов с целью разработки специализированных ИМС, ориентированных на российские линии связи. Причем речь идет о создании высокоскоростных модемов (4800, 9600, 14400 бод).

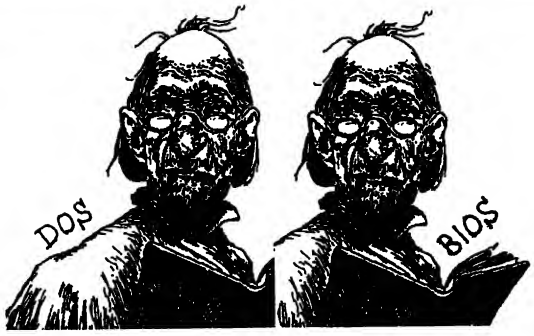
Но если потребность в высокоскоростных модемах назрела, то обращаться в "НАТЕКС" можно уже сейчас. Фирма поставляет проверенные на отечественных линиях связи модемы фирмы ZOOM Telephonics Inc., которая в течение уже трех лет лидирует на рынке модемов США. Надо отметить, что цены, по которым продается аппаратура, выставлены самой фирмой ZOOM Telephonics Inc. и едины во всем мире. Так, внешний модем ZOOM VXV (V.22bis,MNP 5 & V.42bis H/W) вы можете приобрести за $169 ($148 за 100 шт.), а высокоскоростной ZOOM VXV32 (внешний, 9600, V.32,V.22bis, MNP5 & V.42bis H/W с регулировкой выходного уровня сигнала) за $349 ($310). И, наконец, ZOOM VXV32bis — внешний модем с максимальной скоростью работы 14400 бод (MNP5 & V.42 bis, H/W), стоимостью $399 ($349).

Как видите, стоимость аналогичного по характеристикам модема импортного производства выше как минимум на 30%. И дело здесь не в демпинговой политике, ориентированной на продвижение собственной продукции в ущерб западному конкуренту (вы ведь можете сравнить цены на аналогичные модемы у других поставщиков коммуникационного оборудования). Нет, речь идет о грамотной организации разработки и производства высокотехнологичной продукции, которая очень наглядно демонстрирует смысл пословицы: "За морем телушка полушка, да рубль перевоз!". Да и вряд ли "НАТЕКС" будет выпускать некачественную продукцию, обеспечивать ее гарантийным сроком службы, и утешаться низким уровнем цен и низкой нормой прибыли при высоком числе продаж.

Наступление фирмы "НАТЕКС" идет и по всему фронту. Фирма расширяет спектр разрабатываемых и поставляемых изделий, связанных прежде всего с телекоммуникационными технологиями. Так, для поддержки host машин Вы можете приобрести три типа (!) 4-канальных плат RS-232C. Для подключения модемов Вам предложат различные типы кабелей RS-232, а также кабели "нуль-модемов".

Ценовая политика фирмы "НАТЕКС" производит приятное впечатление. Дело не только в том, что модемы собственного производства продаются по более низким ценам, чем импортные. Это объясняется тем, что за валюту фирмой приобретаются комплектующие, а все производство развернуто в России. Интерес представляет политика в области оптовых скидок, ориентированная на максимальное распространение продукции. А в этом случае в выигрыше останутся и фирма, и конечные пользователи.

Звоните по телефону: 8(095)324-40-00, отправляйте факс: 8(095)324-28-07 или письменный запрос по адресу: 103031, Москва, а/я 839, НТЦ "НАТЕКС".



Снова о TSR или как пользоваться функциями DOS и BIOS из Вашей резидентной программы

А. Аброськин

Надеюсь, что для кого-нибудь это послужит отправной точкой для изучения ассемблера и для создания собственных TSR...

Безусловно одной из пугающих задач, с которой сталкивается программист, является создание 100% резидентных программ (Terminate and Stay Resident /окончить и стать резидентной/ — TSR). При использовании языков высокого уровня часто цель становится недостижимой, несмотря на обилие всевозможных библиотек, таких как, например, Turbo Professional, и программист, не использующий языка ассемблера, как правило, вынужден хоронить свою очередную идею. А может, не стоит отчаиваться и попробовать самому программировать на ассемблере? Вы готовы? Тогда сразу в бой!

Структурно TSR программа из двух частей: транзитной и резидентной. Транзитная часть "доставляет и устанавливает" в DOS необходимые процедуры резидентной части программы, передает управление операционной системе, оставляя в памяти резидентную часть, которая, собственно, и будет заниматься творческой деятельностью, определенной фантазией и квалификацией автора программы. Транзитная часть может определять наличие программы в памяти, а также осуществлять выгрузку программы из памяти и восстановление захваченных векторов прерываний.

Рассмотрим поближе организацию резидентной части программы. Как правило, в ее состав входит диспетчер DOS, одна или несколько PopUp программ со своими наборами данных.

PopUp программа — это исполняемый код для выполнения сервисной функции, ради которой собственно и пишется TSR.

Диспетчер DOS предназначен для слежения за операционной системой с целью определения безопасного состояния, во время которого можно выполнить PopUp программу. Для этого диспетчер должен содержать в себе процедуры, следящие за состоянием MS-DOS, ROM BIOS и аппаратного обеспечения.

Так как TSR программа может быть активизирована в любой момент времени, на плечи диспетчера ложится большая ответственность за принятие решения о передаче управления PopUp программе. Такие сложности связаны, прежде всего, с нереентерабельностью DOS и BIOS, но они вполне преодолимы, поэтому рассмотрим основные моменты, на которые стоит обратить внимание при написании диспетчера DOS.

Естественно, что если Ваша PopUp программа не собирается использовать процедуры BIOS или DOS, то Ваш диспетчер может состоять из пары строк, ну а если объектом PopUp является дисковый файл или устройство, то придется попотеть.

Как обойти BIOS? Для этого заменим подпрограмму обработки прерывания BIOS на свою, в которой перед вызовом "настоящего" прерывания будем взводить некий флажок, а после отработки прерывания сбрасывать его, ну а диспетчеру поручим проверять этот флажок и принимать соответствующее решение (то есть если флаг будет сброшен, то можно безопасно пользоваться "украденным" прерыванием и наоборот).

Теперь займемся выяснением состояния аппаратных средств. Интерфейс программируемого контроллера прерываний Intel 8259A оказывается весьма кстати. Надо просто спросить у контроллера, не занят ли он другими прерываниями, и если все в порядке, дать добро на выполнение PopUp (подробности описаны далее).

Осталось окинуть взглядом MS-DOS и приступить к основной работе. Состояние DOS можно получить по адресу DosMode, который при инициализации TSR программы можно получить

при помощи недокументированной функции DOS Int34H ("Дать адрес статуса реентерабельности DOS"), и сохранить его в DosMode для последующих обращений. Нулевое значение флага говорит о том, что DOS находится в безопасном состоянии. Однако Вас может постичь разочарование, так как некоторые программы (в том числе COMMAND.COM) большую часть времени проводят, запросив ввод с клавиатуры через функцию OAh, а на службу DOS наложено табу. Можно выйти из положения, самостоятельно перехватывая Int21H (см. TEACH HELP), или воспользоваться тем фактом, что для выполнения полезной системной деятельности перед вызовом некоторых функций, а также находясь в "холостом" состоянии, DOS вызывает обработчик прерывания Int28H, первоначально указывающий на IRET. Значит можно перехватить вектор Int28H и поручить новой подпрограмме выполнить наши нужды, однако следует помнить, что по окончании всех дел следует отдавать управление старой подпрограмме Int28H.

Кроме перечисленного, диспетчер должен определить, не пытается ли он активизировать PopUp в тот момент, когда она уже получила управление. Сделать это можно с помощью того же флага состояния, который будет выставляться и сбрасываться самой PopUp.

Примерно таким образом (порядок может быть изменен) диспетчер может оценивать окружающую обстановку и в случае успеха вызывать соответствующую PopUp, если при этом PopUp может разрушить какие-либо регистры или управляющие области, то диспетчер обязан сохранить эту информацию и после отработки PopUp восстановить оригинальные значения затертой информации (например, в результате ошибки при работе PopUp с диском может быть потеряна расширенная информация об ошибке прерванной программы).

В приведенном фрагменте (листинг 1) предполагается, что транзитная часть TSR программы уже переустановила векторы 05h, 1Bh, 10h, 13h,23h и 24h на свои обработчики прерываний, которые изменяют свои флаги в соответствии с принципами, описанными ранее, а сами процедуры являются резидентной частью TSR программы. Кроме этого следует отметить, что если Вас не интересует содержимое вектора 05h, то получить статус печати копии можно, получив непосредственно ключевой байт в области данных BIOS (0:0500).

Можно отметить, что хотя в документации по DOS и говорится о нереентерабельности BIOS, эксперименты показали, что некоторые процедуры являются повторно входимыми, например, отдельные функции Int10h; опираться на такую информацию рискованно, но об этом не мешает знать. О том, можно ли использовать какие-либо функции DOS или BIOS, Вы узнаете по фатальному сбою системы.

Ну а теперь настало самое время привести пример работающей TSR программы и посмотреть некоторые практические приемы, которые используются при решении подобных задач.

Программа HACKVIEW (листинг 2) представляет собой более сложный объект, нежели RESET, здесь реализованы выполнение дисковых операций, процедуры обработки ошибок и некоторые другие детали.

HACKVIEW — резидентная программа сохранения экрана, пишет копию экрана через заданный промежуток времени в дисковый файл.

Идея этой программы возникла у меня в то время, когда на моем горизонте появился пользователь, интересующийся в мое отсутствие содержимым винчестера. HACKVIEW как нельзя лучше определила и зафиксировала круг его интересов, правда, для этого мне пришлось уменьшить интервал до 30 секунд и найти несколько лишних мегабайтов, но, как говорится, игра стоила свеч.

Можно найти и другие применения HACKVIEW, например для "снятия" последовательности действий какой-либо "быстрой" программы, или использовать ее как основу для резидентного анализа содержимого экрана (например, при проверке правописания), можно, наконец, использовать ее в качестве шутки для своего приятеля-программиста, скажем, если он пишет свои программы на Pascal, то, посмотрев на экран и найдя там знаменитые Begin или End, Вы можете выразить свое бурное удовольствие по этому поводу, инвертировав их изображение и дописав рядом с ними комментарии типа {Hello, Begin!} или {Good By, End!}, остальное подскажет ваша фантазия.

Заключительный совет тем, кто до него добрался, — для обеспечения "живучести" как диспетчер, так и сама программа не должны содержать ошибок. Ошибки во время исполнения будут иметь наивысший приоритет, что несомненно приведет к смертельному исходу, платой за который в лучшем случае будет нажатие кнопки "RESET", ну а в худшем при работе с прерываниями типа Int13H — испорченное настроение окружающих и кропотливая работа по восстановлению утраченных данных.

Если верить специалистам Borland, то "...написание и отладка процедур прерывания напоминают карабканье по острым склонам, а разработка профессиональных TSR сравнима со скольжением по леднику. Но как показывает разнообразие резидентных демонстрационных программ, Вы можете легко справиться с этой гигантской задачей...".

И действительно — не ошибается тот, кто ничего не делает, иными словами: не бойтесь ошибиться, изучайте функциональные примеры, будьте терпеливы и настойчивы, пробуйте, экспериментируйте, всегда интересуйтесь тем, что Вы делаете, и Вы будете вознаграждены поистине профессиональными результатами, которыми бы гордился любой программист мирового класса. Удачи! ■

Рекомендуемая литература
The MS-DOS Encyclopedia.- Microsoft Press, 1988.

Листинг 1

```
;-------------------------------------------------
; Может вызываться по прерыванию от таймера (1Ch)
;-------------------------------------------------

DISP  PROC  FAR             ; Фрагмент кода диспетчера
                            ; в ожидании DOS
.....                       ;
      call  DOSstatus       ; Получить состояние DOS
      call  INTstatus       ; Получить состояние BIOS
      xor   al,al           ;
      cmp   al,cs:FlagINT   ;
      jne   exit            ; Если не стабильно - возврат

      cmp   al,cs:FlagDOS   ; Если DOS нереентерабельна,
      jne   exit            ; то выход из TSRprint,
      cmp   al,cs:FlagTSR   ; TSR активна ?
      jne   exit            ; Да, — возврат,
      call  PopUp           ; иначе активизируем PopUp
exit:
.....
.....
.....
DISP  ENDP

;-------------------------------------------------
; Область данных резидентной части TSR программы
;-------------------------------------------------
                            ;
FlagINT     db    0         ; Флаг состояния BIOS
FlagDOS     db    0         ; Флаг состояния DOS
FlagTSR     db    0         ; Флаг активности PopUp
                            ;
DosMode     dd    ?         ; Адрес флага реентерабельности MS-DOS
;

INTstatus   proc  near      ; Возвращает флаг состояния
                            ; Установлен, если BIOS или
                            ; аппаратные средства нестабильны
      push  ax              ; Сохраняем регистр AX
;-------------------------------------------
; Проверяем состояние прерываний
; контроллера прерываний Intel 8259A
;-------------------------------------------
      mov   ax,0000101b     ; AH = 0, AL = OCW для Intel 8259A
      out   20h,al          ; Запрашиваем "in-service"
                            ; регистр 8259A
      jmp   short chk       ;
chk:  in    al,20h          ; Проверяем состояние прерываний
                            ; текущего состояния 8259A
                            ; (единичный бит, если "in-service")
      cmp   ah,al           ;
      jc    exit_INT        ; Возврат, если аппаратное прерывание

;-------------------------------------------
; Проверяем состояние прерываний
; ROM-BIOS
;-------------------------------------------
                            ;
      mov   al,cs:Flag05    ; Проверяем флаг INT05h:
      cmp   al,0            ; если не сброшен,
      jne   exit_INT        ; возврат
      mov   al,cs:Flag1B    ; Проверяем флаг INT1Bh:
      cmp   al,0            ; если не сброшен,
      jne   exit_INT        ; возврат
      mov   al,cs:Flag10    ; Проверяем флаг INT10h:
      cmp   al,0            ; если не сброшен,
      jne   exit_INT        ; возврат
      mov   al,cs:Flag13    ; Проверяем флаг INT13h:
      cmp   al,0            ; если не сброшен,
      jne   exit_INT        ; возврат
      mov   al,cs:Flag23    ; Проверяем флаг INT23h:
      cmp   al,0            ; если не сброшен,
      jne   exit_INT        ; возврат
      mov   al,cs:Flag24    ; Проверяем флаг INT24h:
      cmp   al,0            ; если не сброшен,
      jne   exit_INT        ; возврат
      mov   FlagINT,0       ; Сбросить флаг состояния
      jmp   short exint     ; и выйти из процедуры
exit_INT:
      mov   FlagINT,1       ; Установить флаг состояния
      jmp   short exint     ; и выйти из процедуры
exint:
      pop   ax              ; Восстановить AX
      ret
INTstatus   endp

DOSstatus   proc  near      ; Возвращает: флаг состояния
                            ; установлен, если MS-DOS
                            ; нереентерабельна
      push  ds              ;
      push  dx              ; Сохраняем регистры
      push  ax              ;

;-------------------------------------------
; Проверяем реентерабельность MS-DOS
;-------------------------------------------
      lds   bx,cs:DosMode   ; Загружаем в AL
      mov   al,[bx]         ; флаг активности DOS
      xor   bl,bl           ;
      cmp   bl,al           ; Возврат из процедуры, если
      jne   exit_DOS        ; система нестабильна,
      mov   FlagDOS,0       ; иначе устанавливаем флаг
      jmp   short exdos     ; реентерабельности DOS
exit_DOS:
      mov   FlagDOS,1       ; Устанавливаем флаг
      jmp   short exdos     ; нереентерабельности DOS
exdos:
      pop   ax              ;
      pop   dx              ; Восстанавливаем регистры
      pop   ds              ;
      ret
DOSstatus   endp
```

Листинг 2

```
;
;TASM HACKVIEW.ASM
;TLINK /T/X HACKVIEW.OBJ
;
;P.S. Для более незаметного подключения программы рекомендую
;в Вашей (!) версии убрать все сообщения и загружать
;программу из CONFIG командой install = ...\hv.bin
;
;(hv.bin —> copy/b hackview.com hv.bin)
;
;

INCLUDE MACRO.21
code        segment
            assume    cs:code,ds:code
            org       100h

start:      jmp   install           ; Инициализация
                                    ;
presence    dw    06cEh             ; Ключевое слово присутствия
                                    ; программы в памяти
                                    ;
int08_entry proc  far               ; Обработчик прерывания 08H
                                    ;
            pushf                   ;
            cmp   cs:minutes,1092   ;
            jne   exit              ;
            test  cs:gate08,1               ; gate08 закрыты?
            jnz   exit08                    ; Да, вызываем старый int 08h
```

```
                popf                    ;
                pushf                   ;
                call    dword ptr cs:[old08o]  ; Отрабатываем настоящий int 08h
                mov     byte ptr cs:gate08,1   ; Закрываем ворота
;-------------------------------------------------------------------------
                call    CopyBuffer      ; Пишем в буфер
;-------------------------------------------------------------------------
                mov     byte ptr cs:gate08,0   ; Открываем ворота
                iret                    ; и выдаем IRET
                                        ;
int08_entry     endp                    ;

exit:           inc     cs:minutes      ; Увеличиваем счетчик импульсов
                                        ;таймера  на единицу
exit08:                                 ;
                popf                    ;
                db      0EAh            ; far jump на old int 08h
old08o          dw      0               ; Старый адрес прерывания 08h -
old08s          dw      0               ; смещение и сегмент
                                        ;
;-------------------------------------------------------------------------
gate08          db      0               ; Открываем ------> 0
gate28          db      0               ; Открываем ------> 0
;-------------------------------------------------------------------------

CopyBuffer      proc    near            ; Копируем видеобуфер
                                        ;
                push    ax
                push    bx
                push    cx
                push    dx
                push    ds              ;
                push    es              ; Сохраняем регистры
                push    di              ;
                push    si
                xor     ax,ax
                mov     es,ax           ; По адресу 0:0449 считываем
                mov     al,es:[0449h]   ; ключевой байт текущего видео-
                cmp     al,7            ; режима (то же, что и INT10h)
                ja      short copyend   ; Если графический режим - выход
                jl      short mode ;
                mov     ax,0B000h       ; Адрес b/w TextMode 80x25
                jmp     short setup
mode:
                cmp     al,3
                ja      short copyend   ; Если графический режим - выход,
                je      short text      ; если текстовый - новый адрес
                cmp     al,2            ;
                jne     short copyend   ; Текстовый режим 40x25
text:                                   ;
                mov     ax,0B800h       ; Стартовый адрес TextMode 80x25
setup:
                mov     ds,ax           ; Передаем видеосегмент в DS
                xor     dx,dx
                mov     al,160          ; Готовим необходимые регистры
                mul     dl              ; для корректной работы CopyBuffer
                mov     si,0
                shl     si,1
                add     si,ax
                add     si,es:[044Eh]   ; Добавляем в SI смещение текущей
                mov     ax,si           ; видеостраницы (слово адреса 0:044E)
                push    cs
                pop     es
                lea     di, byte ptr es:[Buffer]
                mov     bx,25
                mov     dx,80       ; В BX число строк, в DX - длина строки
                cld                 ; Очищаем флаг направления
                jmp     short live
memadd:
                add     ax,160      ; Добавляем смещение для следующей строки
                mov     si,ax       ; и передвигаем указатель Buffer
                inc     di          ; на 2 позиции вперед
                inc     di          ;
live:                               ;
                mov     cx,dx       ;
again:          movsb               ; Пересылаем символ
                inc     si          ; Пропускаем атрибут
                loop    again       ; Повторяем
                push    ax
                push    cx
                push    di
                push    si
                push    es          ; Сохраняем регистры
                push    ds          ; для возврата в цикл
                push    cs          ; считывания строк
                pop     ds
                xor     ax,ax
                mov     si,ax
                lea     si,byte ptr ds:[EOL]
                mov     cx,2
                rep     movsb       ; Вставляем <EOL>
                pop     ds          ; Восстанавливаем
                pop     es          ; регистры и
                pop     si
                pop     di
                pop     cx
                pop     ax
                dec     bx          ; уменьшаем счетчик строк
                jne     memadd
                mov     cs:FlagBUF,1    ; Взводим флаг готовности
                mov     cs:minutes,0    ; Обнуляем счетчик импульсов таймера
copyend:
                pop     si
                pop     di
                pop     es
                pop     ds
                pop     dx          ; Восстанавливаем
                pop     cx          ; регистры
                pop     bx
                pop     ax
                ret
CopyBuffer      endp

CopyFile        proc    near

                cli                 ;
                in      al,21h      ; Для исполнения критического участка
                or      al,00000001b    ; запрещаем прерывание от системного
                out     21h,al      ; таймера ( бит 0 )
                sti
                mov     ax,3300h    ; Получаем текущее состояние
                int     21h         ; ситуации "break" и сохраняем
                mov     Break,dl    ; его для последующего восстановления
                xor     dl,dl       ; Запрещаем проверку ситуации "break"
                mov     ax,3301h    ; на время ввода-вывода с диска
                int     21h         ;

                push    ds          ;
                xor     bx,bx       ; Получаем расширенную информацию
                mov     ah,59h      ; об ошибке и сохраняем ее для
                int     21h         ; последующего восстановления

                mov     cs:ErrorDS,ds   ;
                pop     ds          ;
                mov     cs:ErrorAX,ax   ;
                mov     cs:ErrorBX,bx   ;
                mov     cs:ErrorDX,dx   ;
                mov     cs:ErrorSI,si   ;
                mov     cs:ErrorDI,di   ;
                mov     cs:ErrorES,es   ;
                                        ;
                mov     ah,51h      ; Получаем адрес PSP
                                    ; текущего процесса
                int     21h
                mov     cs:PPSP,bx  ; BX = PSP текущего процесса
                mov     bx,cs:TPSP  ; BX = резидентный PSP
```

Листинг 2 (продолжение)

```
        mov   ah,50h          ; Устанавливаем адрес PSP
        int   21h             ;
                              ;
        mov   ax,cs           ;
        mov   ds,ax           ;
        lea   dx,File         ;
        mov   ah,3Dh          ; Открываем файл
        mov   al,1            ; (единица) для записи
        int   21h             ;

        jc    open_error      ; Если не существует,
                              ; попробуем его создать
open:
                              ; Сохраняем "file handle"
                              ; в BX
        mov   bx,ax           ;
        mov   ah,42h          ;
        mov   al,2            ;
        mov   cx,0            ;
        mov   dx,0            ; Передвигаем указатель
                              ; в конец файла (в AL - два)
        int   21h

        push  cs
        pop   ds
        lea   dx,Buffer
        mov   cx,2050         ; В CX - число байтов для считывания
        mov   ah,40h          ; Записываем Buffer в File
        int   21h

                              ;
close:                        ;
        mov   ah,3Eh          ; Закрываем файл
        int   21h
        jmp   short exit_28H  ;
                              ;
open_error:                   ;
        mov   ax,cs
        mov   ds,ax
        lea   dx,File         ; Создаем "file handle"
        mov   cx,02h          ; для File с атрибутом
        mov   ah,3Ch          ; "hidden" (CX = 02h)
        int   21h
        jc    short exit_28H  ; Выходим по ошибке
        jmp   short open      ; Переходим к работе с файлом
                              ;
                              ;
exit_28H:                     ;
        mov   bx,PPSP         ; BX = предыдущий PSP
        mov   ah,50h          ; Устанавливаем адрес PSP
        int   21h
                              ;
        mov   dx,offset cs:ExError
        mov   ax,5D0Ah
        int   21h

                              ;
        mov   dl,Break        ;
        mov   ax,3301h        ;
        int   21h
        mov   cs:FlagBUF,0    ; Сбрасываем флажком буфера
exit_28H_end:
        cli
        in    al,21h          ; Разрешаем замаскированиое
        and   al,11111110b    ; ирерывание по таймеру
        out   21h,al          ; (пишем ноль в бит 0)
        sti
        ret
CopyFile  endp
;------------------ DATA ------------------------
File     db    "c:\himage.idx",0      ; Имя файла
Break    db    ?                      ; Флаг проверки "break"
FlagBUF  db    0                      ; Флажок заполнения буфера
Buffer   db    2050 dup (020h)        ; Емкость для hacker's buffer
EOL      db    0Dh,0Ah                ; EOL для строк File
minutes  dw    0                      ;Число импульсов таймера
                                      ; ( 1 мин. = 1092 тика )
PPSP     dw    ?                      ;Захваченный (не ваш) PSP
TPSP     dw    ?                      ; Резидентный (наш) PSP
                                      ;

ExError  label byte                   ; Захваченная расширенная
                                      ; информация об ошибке
ErrorAX  dw    ?
ErrorBX  dw    ?
ErrorCX  dw    ?
ErrorDX  dw    ?
ErrorSI  dw    ?
ErrorDI  dw    ?
ErrorDS  dw    ?
ErrorES  dw    ?
         dw    3 dup (0)
;-----------------------------------------------------------

int28_entry proc  far                 ; Обработчик прерывания 28H
                                      ;
         pushf                        ;
         test  cs:gate28,1            ; gate28 закрыты?
         jnz   exit28                 ; Да, до свидания mr. FAR JUMP
         popf
         pushf                        ;
         call  dword ptr cs:[old28o]  ; Отрабатываем int 28h
         mov   byte ptr cs:gate28,1   ; Закрываем ворота
                                      ;
         PushAll                      ; Save All Regisres
         cmp   cs:FlagBUF,1           ; buffer o'key?
         jne   short @@@              ; No
         call  CopyFile               ; Yes, пишем в файл
@@@:                                  ;
         PopAll                       ;Restore All Registers
;-----------------------------------------------------------
         mov   byte ptr cs:gate28,0   ; Открываем ворота
         iret                         ;
                                      ;
int28_entry endp

exit28:
         popf
         db    0EAh                   ; far jump на old 28h
;
old28o         dw    0                ; Старый адрес прерывания 28h -
old28s   dw    0                      ; смещение и сегмент
                                      ;

;-----------------------
      Инициализация
;-----------------------
install:                              ;
         mov   ax,3508h               ; Получаем адрес
         int   21h                    ; прерывания 08h
         mov   ax,es:[bx-2]           ; Проверяем на присутствие
         cmp   ax,cs:presence         ; программы в памяти
         jnz   load                   ; Нет - переход на load
         jmp   already                ; Да - переход на already

load:
         push  es
         push  ds
         mov   ah,51h
         int   21h
         mov   cs:TPSP,bx             ; Запоминаем наш PSP;
         mov   ax,3508h               ; Получаем адрес
         int   21h                    ; прерывания 08h
         mov   word ptr old08s,es
         mov   word ptr old08o,bx
                                      ;
```

```
                                ;
        lea   dx,int08_entry    ;
        mov   ax,2508h          ; Устанавливаем новый вектор
        int   21h               ; прерывания 08h
                                ;
                                ;
        mov   ax,3528h          ; Получаем адрес
        int   21h               ; прерывания 28h
        mov   word ptr old28s,es
        mov   word ptr old28o,bx
                                ;
                                ;
        lea   dx,int28_entry    ;
        mov   ax,2528h          ; Устанавливаем новый вектор
        int   21h               ; прерывания 28h

        mov   ax,cs             ;
        mov   ds,ax             ; Выводим сообщение
        lea   dx,mes_load       ; об успешной загрузке
        mov   ax,0900h          ; программы
        int   21h

        pop   ds                ; Восстанавливаем регистры
        pop   es                ;
                                ;
        mov   dx,offset install ; Устанавливаем резидентную
        int   27h               ; часть программы

already:
        mov   al,ds:[082h]      ; Разбираем командную строку
        cmp   al,'/'            ; и проверяем на допустимые
        je    key_1             ; ключи
        jmp   no_key
key_1:
        mov   al,ds:[083h]
        cmp   al,'E'
        je    key_2
        cmp   al,'e'
        je    unload
        jmp   bad_key
key_2:
        mov   al,ds:[084h]
        cmp   al,'N'
        je    key_3
        cmp   al,'n'
        je    key_3
        jmp   bad_key
key_3:
        mov   al,ds:[085h]
        cmp   al,'D'
        je    unload
        cmp   al,'d'
        je    unload
        jmp   bad_key


unload:
        mov   ax,word ptr es:old08o ; Получаем адреса для
        mov   dx,word ptr es:old08s ; восстановления вектора INT08h
        mov   cx,es                 ; Сохраняем резидентную
                                    ; CS часть программы
        cli
        xor   bx,bx                 ; Пересылаем ноль
        mov   es,bx                 ; в ES
        mov   bx,08h*4
        mov   es:[bx],ax            ; Восстанавливаем оригинальный
        mov   es:[bx+2],dx          ; вектор INT08h

        mov   es,cx
        mov   ax,word ptr es:old28o ; Получаем адреса для
        mov   dx,word ptr es:old28s ; восстановления вектора INT28h
        xor   bx,bx
        mov   es,bx
        mov   bx,28h*4
        mov   es:[bx],ax            ; Восстанавливаем
оригинальный
        mov   es:[bx+2],dx          ; вектор INT21H
        sti

        mov   ah,49h
        mov   es,cx                 ; Восстанавливаем ES
        int   21h                   ; и освобождаем память

        mov   ax,0003h
        int   10h
        mov   ax,cs
        mov   ds,ax
        lea   dx,unloaded           ;
        mov   ax,0900h              ; Выдаем сообщение об успешной
        int   21h                   ; выгрузке программы
        jmp   finish                ; из памяти

no_key:
        mov   ax,cs                 ;
        mov   ds,ax                 ; Выдаем соответствующее
        lea   dx,loaded             ; сообщение о присутствии
        mov   ax,0900h              ; программы в памяти
        int   21h                   ;
        jmp   finish

bad_key:
        mov   ax,cs                 ;
        mov   ds,ax                 ; Выдаем соответствующее
        lea   dx,error              ; сообщение о неверном
        mov   ax,0900h              ; ключе выгрузки
        int   21h                   ;

finish:
        mov   ax,4C00h              ; Завершаем программу
        int   21h                   ;

mes_load  db  'O"Key! Hacker"s Viewer loaded. ',0Dh,0Ah,'$'
loaded    db  'Hacker"s Viewer already loaded!',0Dh,0Ah,'$'
unloaded  db  'Hacker"s Viewer rewoved from MEMORY.',0Dh,0Ah,'$'
error     db  'For UNLOAD please begin to run with /END or/end.'
          db  0Dh,0Ah,'$'
code      ends
          end start

;-------------------------------------
;       MACRO.21 for HACKVIEW.ASM
;-------------------------------------
PushAll   MACRO
          pushf
          push ax
          push bx
          push cx
          push dx
          push di
          push si
          push bp
          push es
          push ds
          ENDM
;
PopAll    MACRO
          pop  ds
          pop  es
          pop  bp
          pop  si
          pop  di
          pop  dx
          pop  cx
          pop  bx
          pop  ax
          popf
          ENDM
```

Законы надо уважать!

Вспомните, сколько раз в неделю Вы сталкиваетесь с необходимостью разыскать тот или иной правовой акт, типовую форму договора или новую форму бланка (например, для уплаты налога в пенсионный фонд)? А разыскав, пытались вспомнить, не принято ли по этому поводу нашими депутатами, министерствами и так далее чего-нибудь такого, что напрочь отрицает данную бумагу и устанавливает совершенно другие "правила игры". Наконец самый последний, свежий документ обнаружен, а после еще одного (одного ли?) обращения к книжному шкафу найден соответствующий научно-практический комментарий.

Но вот Вы написали необходимый документ, подшили его в дело и ... выбросили из головы всю проделанную работу. В результате когда на Ваше письмо придет ответ, Вам придется повторять весь цикл поиска повторно.

Но в этом месте, после столь драматичного (и реалистичного, смеем Вас уверить) вступления, надо бы грозно нахмурить брови и заявить: "Во всем этом, уважаемые читатели, виноваты Вы сами! Ибо не приобрели до сих пор уникальную восходящую звезду на небосклоне рынка отечественного программного обеспечения — юридическую информационно-справочную систему "ЮСИС", разработанную и поставляемую юридическим информационным агентством Intralex."

"ЮСИС" представляет собой программный комплекс, обеспечивающий поиск и ввод информации о нормативных актах, научно-практическом комментарии к ним, судебной, арбитражной и нотариальной практике применения, типовых формах договоров, бланков и тому подобном, включая ведение делопроизводства.

При этом "ЮСИС" поддерживает две базы данных:
- аннотированная информация о принятых нормативных актах и практике их применения (картотека);
- полные тексты нормативных актов, документов и т.д.

Система предлагает широкие возможности поиска необходимых документов и включает установку критериев поиска по следующим параметрам:
- вид материала (нормативный акт, практика применения, комментарий и так далее);
- раздел по классификатору;
- наименование или контекст наименования нормативного акта;
- орган, принявший нормативный акт;
- вид, дата (период), номер и источник публикации акта;
- ключевые слова, характеризующие акт.

Вы можете задавать значения для одного или для всех параметров одновременно, а найдя необходимый документ, воспользоваться для его просмотра любимым текстовым редактром, например, "Лексиконом" или МЕ.

Но это еще не все! Если Вы помните, в начале статьи мы столкнулись с проблемой увязывания нормативных документов с повседневной деятельностью. Так вот, "ЮСИС" позволяет пользователю создать собственную базу данных как правовой, так и любой другой информации, с установкой всех тех атрибутов, которые используются и в поставляемой Intralex базе (классификатор, орган, принявший документ, ключевые слова). Это означает, что Вы сможете самостоятельно учитывать в "ЮСИС" акты "местного" значения.

Кроме того, "ЮСИС" предоставляет Вам возможность вести учет, регистрацию и поиск входящей и исходящей корреспонденции (делопроизводство в полном объеме), внутренних нормативных документов с установлением ссылок на соответствующие акты, включенные в систему, и практику их применения.

Не выходя из "ЮСИС", Вы можете ознакомиться с необходимым актом, составить документ и зарегистрировать полученный результат в системе с привязкой к использованным нормативным актам. А когда впоследствии Вам потребуется поднять данный документ, то для того чтобы освежить в памяти правовую основу его создания, Вам достаточно будет нажать одну клавишу!

Но дотошный читатель, периодически натыкающийся (при переключении телевизионных каналов) на заседания Верховного Совета, задаст коварный вопрос: "А что делать с новыми документами? С теми, которые будут приняты после покупки "Юсис"? Мне придется вводить их самому или Intralex будет периодически обновлять базу данных?"

Ах, как жаль, что на бумаге не передать ехидной интонации в голосе дотошного читателя! Потому что Intralex и в самом деле обеспечивает ЕЖЕМЕСЯЧНОЕ сопровождение поставляемых баз данных, рассылая новые данные на дискетах.

Но дотошоного читателя так просто не смутить!

— Ну зачем нам такая куча дискет?! Мы народ просвещенный, у нас вот даже и модем есть! Не какой-нибудь там Lexand, а настоящий импортный, говорят, даже Hayes совместимый. Вот если бы по нему новые данные пересылали...

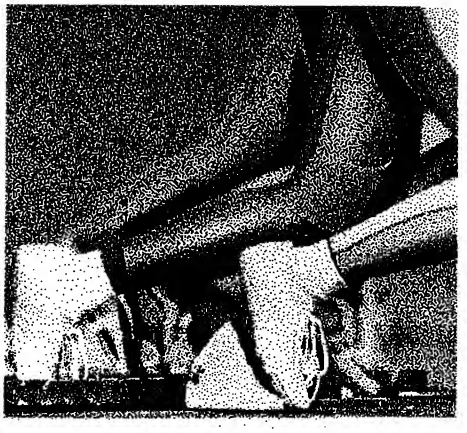
Ну просто одно удовольствие придумывать реплики за дотошного читателя, когда есть чем на них ответить! Потому как, кроме рассылки дискет, Intralex обеспечивает обновление базы данных:
- через собственную BBS (Вы можете получать обновленные версии ЕЖЕДНЕВНО !!!);
- через сеть RELCOM;
- через систему РИЦ в режиме ON-LINE;
- через сеть ТВ-ИНФОРМ-бизнес.

Intralex постоянно совершенствует программное обеспечение комплекса "ЮСИС". Все новые версии поставляются пользователям БЕСПЛАТНО.

И, наконец, еще одно замечание. Почему-то, когда обсуждается сервис и забота о пользователях системы, всегда сравнивают с Западом. Вот, мол, все как у них. Хватит ныть, господа! "ЮСИС" говорит сам за себя — это настоящее РОССИЙСКОЕ КАЧЕСТВО! ■

Адаптер VGA. Режим 256 цветов

Б. Телеснин



Введение

Все адаптеры VGA поддерживают режим 13h с разрешением 320x200 точек и 256 цветов одновременно. К недостаткам этого режима можно отнести использование только одной графической страницы и низкое разрешение. Если Вы решили устранить эти недостатки, Вам не обойтись без некоторых специфических знаний.

В части 1 статьи будет рассмотрено решение проблемы полного использования видеопамяти в 256-цветовых режимах, а также дана весьма полезная информация о формировании изображения на экране в этих режимах.

В части 2 статьи будут рассмотрены возможности создания 256-цветовых режимов с высоким разрешением, а также поддержка расширения видеопамяти до 512 Кбайтов.

Programmer

Если Вы еще не успели попробовать себя в программировании регистров видеоадаптера, Вы можете создать собственный пакет на основе процедур, приведенных в статье.

Если Вы еще не успели попробовать себя в регистровом программировании, Вы все равно можете сделать свой графический пакет, взяв в качестве базы процедуры из демонстрационной программы, приведенной в листинге.

Если Вы все-таки решили разобраться в деталях, Вам будет полезно почитать описание VGA или EGA, чтобы получить начальное представление о том, что такое видеоадаптер, видеопамять, как программировать регистры и так далее.

Автор надеется, что в отсутствие стандартных графических пакетов, работающих в режиме 256 цветов, и более или менее удовлетворительной информации об этих режимах, информация, изложенная здесь, окажется вам полезной.

ЧАСТЬ 1

Итак, для тех, кто не боится слов регистр, контроллер, адрес, сделаем первую пробу — установим режим 13h с четырьмя графическими страницами.

Для этого потребуется следующее.

1) Предварительная установка видеорежима 13h.
2) Использование бита 5 регистра "смешанный вывод", обозначенного XOR_17 (табл. 1), смысл этого обозначения выяснится позже.
3) Установка битов 2-3 регистра "смешанный вывод" контроллера графики (назовем их MemoryMap) в 0.
4) Использование регистра "стартовый адрес" (StartAddress) контроллера электронно-лучевой трубки (ЭЛТ, табл.2).
5) Значение начального адреса видеобуфера для центрального процессора (ЦП).

Таблица 1

Установка на запись/чтение		
Активная страница	Значение XOR 17	Начальный адрес видеобуфера для ЦП
0	1	A000h:0000h
1	1	B000h:0000h
2	0	A000h:0000h
3	0	B000h:0000h

Таблица 2

Установка видимой страницы	
Видимая страница	Значение StartAddress
0	0000h
1	4000h
2	8000h
3	0C00h

Если Вы владеете навыками регистрового программирования, самостоятельная проверка изложенного не займет у Вас много времени. Однако в любом случае Вы сможете запустить демонстрационную программу, написанную на Turbo Pascal 6.0.

Программирование доступа к видеопамяти и формирование изображения

Для начала нам понадобится некоторая информация о формировании видеоадаптером изображения из видеобуфера и о способах адресации ЦП в видеопамять в режиме 256 цветов.

Заметьте, что речь идет о режиме "256 цветов" и при этом не упоминается о разрешении на экране, что означает универсальность приведенных ниже способов для любого (в пределах разумного) разрешения в режиме 256 цветов!

Формирование изображения на экране из видеобуфера в режиме 256 цветов

Работа атрибут-контроллера

В этом разделе мы познакомимся с тем, как происходит формирование 256 цветов в видеоадаптере VGA. Начнем с того, что цвет пиксела на экране формируется в атрибут-контроллере. Работа атрибут-контроллера достаточно сложна, поэтому ограничимся рассмотрением только двух режимов атрибут-контроллера (16 и 256 цветов). Назовем эти режимы условно C16 и C256 (соответственно 16 и 256 цветов).

Входные данные поступают на вход атрибут-контроллера непосредственно от контроллера графики в виде 4 параллельных битовых потоков по 8 битов в каждом. Эти потоки поступают в параллельно-последовательный преобразователь (ППП), который за каждый такт выбирает по одному биту из каждого потока и преобразует их в последовательную форму. После этого происходит побитовый сдвиг каждого потока вправо на один бит. На следующем такте происходят выборка и загрузка следующих 4 битов. Восемь тактов пикселов образуют 1 символьный такт. Символьный такт определяет период времени, по истечении которого контроллер графики снова загружает данные. Столь конкретная информация о временных интервалах потребуется нам при установке режимов высокого разрешения.

В режиме C16 4 бита на выходе ППП непосредственно определяют индекс одного из регистров палитры (индексы 0..15), который совместно с регистром выбора цвета определяет 8-битовый номер регистра цвета (0..255) цифро-аналогового преобразователя (ЦАП) (рис. 1).

В режиме C256 первые 4 бита на выходе ППП образуют младшую половину, а следующие 4 бита — старшую половину байта, который определяет номер регистра цвета ЦАП (рис. 2). Регистры палитры в этом режиме не используются.

Режимы C16 и C256 устанавливаются в регистре управления режимов (индекс 10h) атрибут-контроллера в бите 6 (AtributeColorMode). При установке этого бита в 0 получаем режим C16, в 1 — C256. Отметим, что установка в AtributeColorMode=1 является необходимым и практически достаточным условием для получения 256 цветов одновременно. Это значит, что после инициализации ЛЮБОГО (!!!) 16-цветного графического режима достаточно только установить AtributeColorMode=1 и Вы получите 256 цветов. В дальнейшем изложении подразумевается, что используется режим C256.

Загрузка параллельно-последовательных преобразователей

Теперь рассмотрим процесс посылки данных контроллером графики на вход атрибут-контроллера. Этот процесс в литературе можно встретить под названием "загрузка параллельно-последовательных преобразователей". За один символьный такт происходит выборка 4 байтов видеопамяти с одинаковыми адресами (по байту из каждой плоскости) и их загрузка в ППП. Существуют два режима загрузки, назовем их L1 и L2. Возможен третий режим — режим эмуляции CGA, но в нем используются только два потока (для получения 4 цветов), и ввиду невозможности получения 256 цветов этот режим далее не рассматривается.

В режиме L1 байт из плоскости 0 копируется в поток 0, байт из плоскости 1, соответственно, в поток 1 и так далее (рис.3). То есть номер битовой плоскости, в которой находился байт данных, опре-

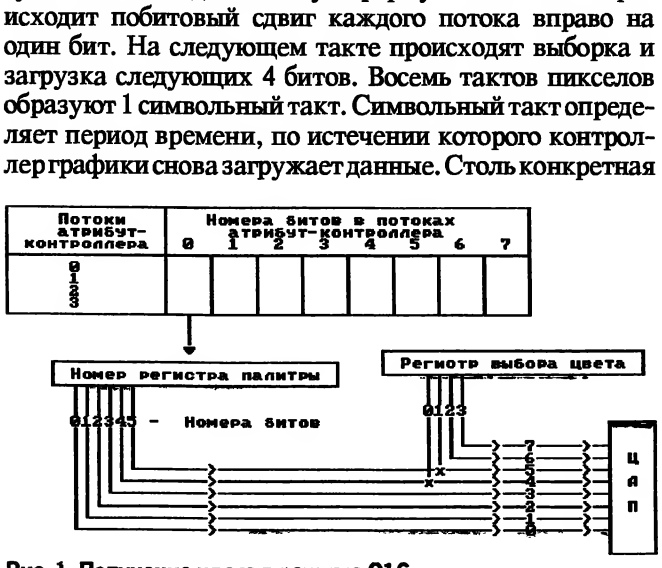


Рис. 1. Получение цвета в режиме C16

Рис. 2. Получение цвета в режиме C256

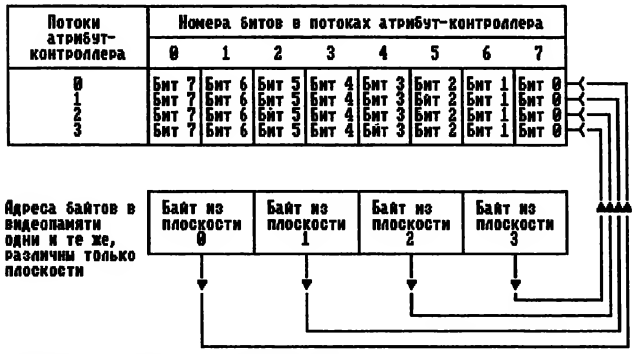


Рис. 3. Загрузка ППП в режиме L1

деляет номер потока, в который этот байт будет записан. При этом, однако, выполняется операция зеркального отражения, в результате которой бит 7 меняется местами с битом 0, бит 6 с битом 1, бит 2 с битом 5 и так далее.

Режим L2 куда более сложен, но именно этот режим наиболее пригоден для работы в режимах 256 цветов. В этом режиме байт из плоскости 0 попадает во все потоки и помещается в битах 0 и 1 каждого потока, байт из плоскости 1 попадает, соответственно, в биты 2, 3 и так далее. При этом младшие 4 бита попадают в порядке возрастания в четные, а старшие 4 бита — в нечетные биты потоков (рис. 4). Введем обозначения:

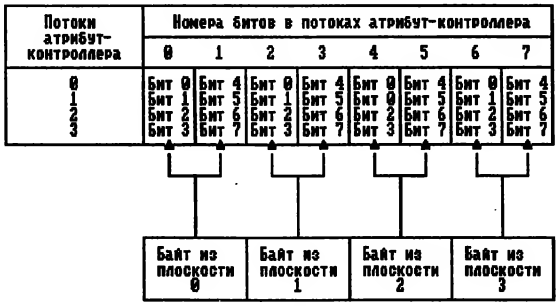


Рис. 4. Загрузка ППП в режиме L2

- Plane — номер битовой плоскости;
- BitPlane — номер бита в байте битовой плоскости;
- Stream — номер потока ППП;
- Bitstream — номер бита в потоке ППП.

Тогда имеем:

Stream = BitPlane mod 4;

(mod — операция взятия по модулю);

BitStream = Plane * 2 + BitPlane div 4;

(div — целочисленное деление).

Покажем теперь, как надо запрограммировать контроллер графики, чтобы выбрать один из режимов - L1 или L2. Для этого потребуется использовать только один бит регистра "режим" (индекс 5h) контроллера графики, а именно - бит 6 "режим 256 цветов" (GRAF_256_Mode). Если GRAF_256_Mode=0, то установлен режим L1, иначе (GRAF_256_Mode = 1) установлен режим L2. Отметим в конце, что режим L2 существует специально для поддержки 256 цветов. Установка этого режима существенно облегчает понимание соответствия "байт видеобуфера — пиксел" (рис.5). Для более наглядного сравнения приводится рис. 6 (режим L1).

Вы можете использовать режим L1, однако в дальнейшем изложении отдается приоритет режиму L2.

Выборка байтов из видеобуфера

Рассмотрим теперь некоторые возможности выборки байтов из видеобуфера, которые предоставляет контроллер ЭЛТ. Выборка начинается с адреса (в данном контексте под адресом нужно понимать лишь 16-битовое смещение), указанного в регистре StartAddress контроллера ЭЛТ, при каждом символьном такте значение адреса увеличивается на 1, и так до определенного значения (на этом сейчас не будем останавливаться). Казалось бы, все просто, но этот адрес не обязательно равен физическому адресу байта в видеопамяти. Нас будет интересовать ситуация, когда программируется циклический сдвиг адреса влево.

Возможен сдвиг на:

0 битов: нормальный режим адресации значения счетчика адреса соответствует физическому адресу (режим S0) (рис.7);

1 бит : умножение адреса на 2-й и 15-й биты замещает бит 0 результата (режим S2) (рис.8);

2 бита : умножение адреса на 4 и 15 бит замещает 1 бит результата, а 14 бит замещает 0 бит (режим S4) (рис.9).

Как это выглядит практически, показано на рисунках 7, 8, 9.

BIOS (при установке видеорежимов) по умолчанию использует в текстовых, графических и 256 цветовых режимах сдвиг на 1 (S2), 0 (S0), 2 (S4), соответственно.

Теперь остаются мелочи — нужно знать, как запрограммировать сдвиг. Для этой цели используются следующие регистры контроллера ЭЛТ (табл.3):

- регистр положения подчеркивания (индекс 14h), бит 6 (DubleWordMode);
- регистр управления режимом (индекс 17h), бит 5 (AddressConvolution) и бит 6 (ByteorWordMode).

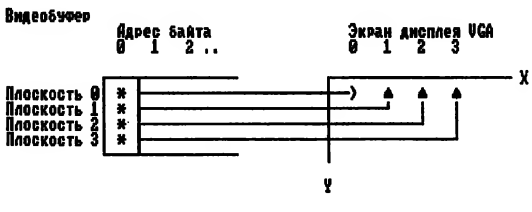


Рис. 5. Формирование пикселов на экране (режимы C256, L2)

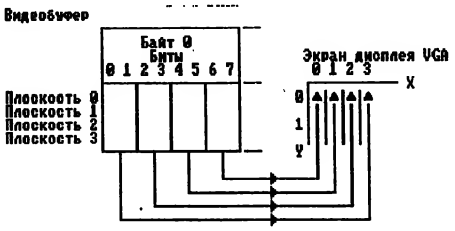


Рис. 6. Формирование пикселов на экране (режимы C256, L1)

Таблица 3

Программирование регистров контроллера ЭЛТ для режимов вращения адреса (S0, S1, S2)

	DoubleWordMode	AddressConvolution	ByteWordMode
S0	0	*	1
S2	0	1	0
S4	1	*	0

Примечание к табл. 3. Знак * здесь показывает, что значение соответствующего бита не играет роли.

Адресация ЦП в видеобуфер в режиме 256 цветов

Для начала необходимо четко уяснить одну вещь: не существует другого способа доступа к видеобуферу, только через видеоконтроллер. Контроллер памяти на IBM PC/AT не производит дешифрацию адресов с A000:0 по B000:FFFF, эта функция принадлежит адаптеру дисплея. Если вам кто-нибудь скажет, что можно обращаться к видеобуферу через DMA, а не через VGA (бывает, и не такое говорят), лучше бросьте в него камень.

Итак, нам в режиме 256 цветов понадобятся только два режима доступа:

- цепочка 4 D4;
- режим произвольного доступа D4.

Рассмотрим их подробнее.

Programmer

Не существует другого способа доступа к видеобуферу, кроме доступа через видеоконтроллер

Режим доступа D4

В этом режиме в формировании реального адреса байта видеобуфера участвуют биты 0, 1, 16, 17 (АП0, АП1, АП16, АП17) адреса ЦП, биты со 2-го по 15-й передаются без изменений. Со стороны адаптера в этом процессе участвуют:

- бит XOR_17;
- 2 бита MemoryMap (их нужно установить в 0, чтобы разрешить использование A000h и B000h);
- бит 3 регистра режима памяти (индекс 4) контроллера синхронизации (носит название "цепочка 4", должен быть установлен в 1) (CHAIN_4).

Итак, рассмотрим формирование адреса байта видеобуфера в режиме D4 (рис.10).

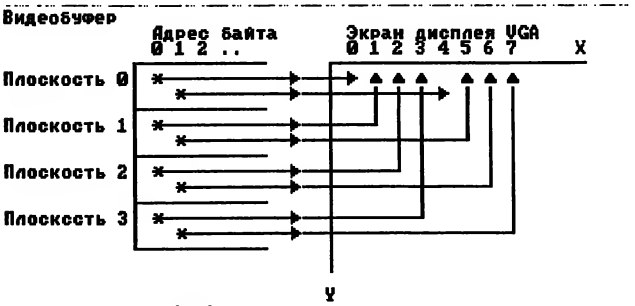


Рис. 7. Сдвиг на 0 (S0)

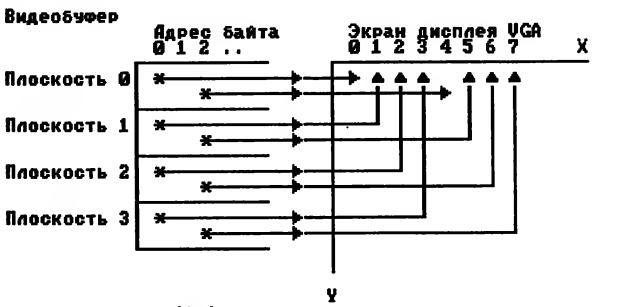


Рис. 8. Сдвиг на 1 (S2)

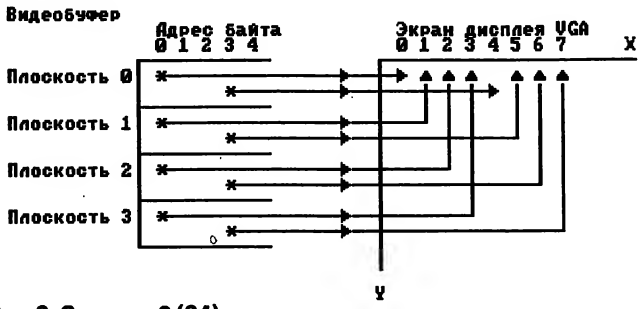


Рис. 9. Сдвиг на 2 (S4)

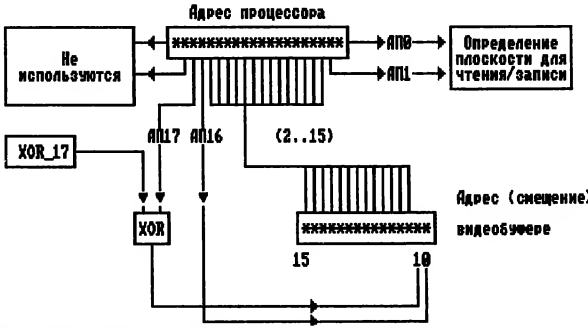


Рис. 10. Формирование адреса в режиме D4

Биты АП0,АП1 определяют номер плоскости видеобуфера, с которой будет произведена операция чтения/записи.

<Номер плоскости = АП0 + АП1 * 2>;

Бит АП16 помещается на место бита 0 адреса байта в выбранной плоскости.

Новое значение бита 1 адреса в плоскости есть результат операции логического отрицания с операндами АП17 и XOR_17.

Биты со 2-го по 15-й передаются в новый адрес без каких-либо изменений.

На рис.11 показано, как записать 4 последовательных байта в каждую из четырех плоскостей.

Отметим два существенных момента:

- регистр маскирования плоскости для записи влияет только на запись данных обычным образом;
- операции чтения/записи могут выполняться как с байтами, так и со словами.

Плоскость	Физические адреса байтов в плоскостях				
	0	1	2	3	4
0	A000:0	B000:0	A000:0	B000:0	A000:4
1	A000:1	B000:1	A000:1	B000:1	A000:5
2	A000:2	B000:2	A000:2	B000:2	A000:6
3	A000:3	B000:3	A000:3	B000:3	A000:7
Значение XOR_17	1		0		1

Рис. 11. Адресация видеопамяти в режиме D4

Примечание к рис. 11. Адрес ЦП каждого байта схематически помещается на его место.

Инициализация этого режима требует:
- CHAIN_4 = 1;
- MemoryMap=0 (по умолчанию BIOS использует MemoryMap=1).

Режим доступа D0

В этом режиме адресация начинается с сегмента A000. Если установлено 256K видеопамяти, то следует установить MemoryMap=1 (установка в 0 ничего не изменит), и Вы получите доступ ко всей видеопамяти, имея 64K адресного пространства.

Если же Вы имеете 512K видеопамяти, необходимо установить MemoryMap=0, и дополнительные 256K будут адресоваться через B000 (еще плюс 64K адресного пространства). Необходимо отметить, что доступ к дополнительной памяти возможен только в режиме D0 (на сегодняшний день другой способ автору неизвестен). Во всяком случае, в режиме доступа D0 наличие дополнительной памяти ничего не меняет, поэтому забудем об этом на некоторое время (об этом придется вспомнить при программировании режимов 256 цветов с высоким разрешением). В этом режиме доступа 16 младших битов адреса процессора непосредственно определяют смещение байта в видеопамяти.

Для использования режима доступа D0 можно ограничиться программированием всего двух регистров:
- регистра маскирования растра контроллера синхронизации (MapMask) (индекс 2, значащие биты 0..3) (табл.4);
- регистра выбора схемы чтения контроллера графики (SelectPlane) (индекс 4, значащие биты 0..1) (табл.5).

Установка этого режима требует:
- CHAIN_4 = 0;
- MemoryMap = 0 или 1.

Таблица 4

SelectPlane	
Биты 1,0	Чтение плоскости
00	0
01	1
10	2
11	3

Таблица 5

MapMask		
Бит	Разрешение записи	Запрещение записи
0	1	0
1	1	0
2	1	0
3	1	0

Синтез режимов доступа и формирования изображения

Теперь нужно совместить способы адресации ЦП и способы формирования изображения так, чтобы получить нечто удовлетворительное. Среди всех комбинаций достаточно использовать только две:
- D4 и S4, назовем этот режим R4 (табл.6 а,б);
- D0 и S0, назовем этот режим R0 (табл.7 а,б).

Рассмотрим теперь, какие возможности мы получим, выбрав R0 или R4 после установки 13h с разрешением 320*200 точек.

Таблица 6а

Установка на запись/чтение		
Страница	Значение XOR_17	Начальный адрес видеобуфера для ЦП
0	1	A000:0000 (64K)
1	1	B000:0000 (64K)
2	0	A000:0000 (64K)
3	0	B000:0000 (64K)

Таблица 6б

Установка видимой страницы	
Страница	Значение StartAddress
0	0h
1	4000h
2	8000h
3	0C000h

Примечание к табл. 6а. В скобках дан размер адресного пространства графической страницы

Соответствие адреса ЦП точкам на экране имеет следующий вид:

Адрес	X	Y
X000h:0	0	0
X000h:1	1	0
X000h:10	10	0
X000h:320	0	1

X000h - это либо A000h, либо B000h в зависимости от нужной страницы (смещение десятичное).

Режим R0

Таблица 7а

Установка на запись/чтение	
Страница	Начальный адрес видеобуфера для ЦП
0	A000:0000 (16K)
1	A400:0000 (16K)
2	A800:0000 (16K)
3	AC00:0000 (16K)

Таблица 7б

Установка видимой страницы	
Страница	Значение StartAddress
0	0h
1	4000h
2	8000h
3	0C000h

Соответствие адреса ЦП точкам на экране имеет следующий вид:

Адрес	X	Y
AX00h:0	0...3	0
AX00h:1	4...7	0
AX00h:10	40...43	0
AX00h:80	0...3	1

AX00h это либо A000h, либо A400h, либо A800h, либо AC00h, в зависимости от нужной страницы (смещение десятичное).

Сравнение режимов R0, R4

В этом разделе автор хочет заранее ответить на вопросы, которые могут возникнуть. Прежде всего выясним, почему бы не использовать только один режим? Установкой режима R4, например, мы значительно упрощаем себе работу с видеобуфером. На языке Turbo Pascal чтение и запись пиксела в режиме R4 — это просто обращение к двумерному массиву с элементами типа byte (смотрите листинг, процедуры PutPixel и GetPixel). Однако все не так просто. Действительно, стоит установить режим с высоким разрешением (например 640*400), и преимущество режима R4 (а именно - простота соответствия байт-пиксел) сойдет на нет, потому что, прежде чем записать точку, придется каждый раз устанавливать сегмент видеобуфера заново. Если Вы установили дополнительную видеопамять (512 Кбайтов), то Вы даже не сможете к ней обратиться в режиме R4. И все же в режиме R4 есть своя прелесть. Если Вы были наблюдательны, то должны были заметить, как выделяется R4 среди всех графических режимов. Он чем-то напоминает по своей простоте текстовые режимы. Полностью отпадает необходимость использования всяких масок, регистров защелок, задвижек и режимов чтения/записи. Непопулярность режима BIOS 13h объясняется, на мой взгляд, отсутствием поддержки 4 страниц со стороны BIOS, а также практически полной неясностью в вопросе установки активной страницы и доступа ко всем 256 Кбайтам памяти. Автор надеется, что теперь этот пробел восполнен. И все же, если у вас возникают проблемы с использованием режима R4, остается только установить R0. К достоинствам режима R0 можно отнести его несомненную универсальность и то, что работа в этом режиме более похожа на работу в 16-цветовых режимах. Последнее относится к тем, кто уже имеет опыт программирования EGA и VGA в 16-цветовых режимах.

Заключение

В листинге дана демонстрационная программа, написанная на языке Turbo Pascal 6.0. Программа показывает использование режимов R0 и R4 для заполнения видеостраниц и прокрутки их на экране в каждом из режимов. Программа также содержит некоторые процедуры ввода/вывода примитивов (PutPixel, GetPixel, Fillscreen), сервисные процедуры (SetActivePage, SetVisualPage, visual_1). Эта программа может послужить неплохой основой для создания вашей собственной библиотеки.

Еще раз хочется напомнить, что в части 2 статьи будет рассказано о том, как установить режим высокого разрешения с 256 цветами. Просьба заранее приготовиться к полному перепрограммированию видеоадаптера. Без сомнения, все, что Вы уже узнали, останется в силе. ■

Анонс

"Монитор" — версия 2.93

В. Аношин. Что же там, в загрузочном секторе?

Детальное описание процедуры загрузки операционной системы. Приведен текст программы-загрузчика на языке ассемблера.

А. Ермолов. Программа-библиотека на PAL

Описание диалоговой программы поддержки библиотеки процедур СУБД "Парадокс". Приведен исходный текст библиотекаря на языке PAL

А. Колесов, О. Павлова. Профессиональная работа в системе Microsoft Quick Basic. Часть 1. Технология разработки программ

Первая часть цикла статей о Quick Basic. Описаны структуры Basic программы, форматы программных модулей, общая схема разработки программ, использование библиотек подпрограмм.

Д. Пластинин. Многооконный интерфейс в среде MS Windows

Обсуждение проблем, связанных с разработкой MDI в среде Windows. Примеры реализации на языке C++ 3.1.

В. Плотников. Программирование каналов ПДП

Описана методика использования контроллера ПДП i8237a при организации обмена данными с периферийными устройствами. Приведены примеры процедур на языке ассемблера.

О. Шарапов. Программная русификация матричных принтеров

Вариант русификации принтеров с различной внутренней организацией. Приведен пример русификатора для принтера Star LC-10, реализованный на языке ассемблера.

М. Трофимов. Кто быстрее?

Статья посвящена вопросам оптимизации скорости исполнения программ.

Журнал "Монитор" приглашает к сотрудничеству:

- Дилеров по распространению журнала
- Специалистов по анализу состояния компьютерного рынка
- Авторов оригинальных статей и книг по программированию и вычислительной технике

Тел. (095) 238-99-33.
Факс (095) 238-32-78.
Адрес для писем: 117419, Москва, а/я 765.

С целью предоставления владельцам персональных компьютеров информации о фирмах, действующих на компьютерном рынке, и предлагаемых ими продуктах и услуга, "Монитор" публикует на страницах журнала каталоги фирм и программ.

"Монитор" публикует также объявления фирм, желающих найти высококвалифицированных специалистов по компьютерам и программированию, и объявления частных лиц, ищущих работу в этой области.

Для получения более подробной информации Вы можете позвонить по телефону (095) 238-99-33 либо выслать открытку со своими координатами по адресу: 117419. Москва, а/я 765.

Листинг

```
program check13;
{Программа демонстрирует различные методы доступа к видеопамяти
и вывода на экран в режимах 256 цветов (320*200 точек).}

const
     Mode_D0 = 0; { режим доступа D0 }
     Mode_D4 = 2; { режим доступа D4 }
     Mode_S0 = 0; { режим сдвига S0 }
     Mode_S2 = 1; { режим сдвига S2 }
     Mode_S4 = 2; { режим сдвига S4 }
     Mode_R0 = 0; { режим R0 }
     Mode_R4 = 2; { режим R4 }
     Kb      = 1024;
     SizePage_R0 = 320 * 200 div 4 ;  { размер памяти на одну страницу
                                        в режиме R0 }
     SizePage_R4 = 320 * 200;         { размер памяти на одну страницу
                                        в режиме R4 }
     Mode_256 : byte = mode_R4;       { текущая установка ( R0 или R4 ) }

type { типы данных для обращения к видеопамяти }
     Screen4Type = array [ 0..199, 0..319 ] of byte; { в режиме R4 }
     Screen0Type = array [ 0..199, 0..79 ] of byte; { в режиме R0 }
     Screen4PtrType = ^Screen4Type;
     Screen0PtrType = ^Screen0Type;

Procedure Cli;
     Inline ( $0FA ); { этот макрос запрещает прерывания }

Procedure Sti;
     Inline ( $0FB ); { этот макрос разрешает прерывания }

{ Процедура SetBIOSMode устанавливает видеорежим BIOS,
номер режима передается в "Mode" }
Procedure SetBIOSMode ( Mode : byte ); assembler;
     asm
     mov al, Mode
     xor ah, ah
     int 10h
     end;

{ Процедура SetSinchronReg устанавливает один из регистров
контроллера синхронизации.
New_v - новое значение, которое будет записано в регистр.
Allow - маска записи в регистр ( будут записаны только те биты
            из New_v, которые установленны в Allow ).
Number - индекс регистра. }
Procedure SetSinchronReg ( New_v, Allow, Number : byte );
     var
          Value : byte;
     Begin
          Cli;
          Port[$3C4] := Number; Value := Port[$3C5];
          Value := ( Value and ( not Allow ) ) or ( New_v and Allow );
          Port[$3C4] := Number; Port[$3C5] := Value;
          Sti;
          End;

{ Процедура устанавливает один из регистров контроллера ЭЛТ.
Входные параметры такие же, как и у процедуры SetSinchronReg }
Procedure SetEltControl ( New_v, Allow, Number : byte );
     var
          Value : byte;
     Begin
          Cli;
          Port[$3D4] := Number; Value := Port[$3D5];
          Value := ( Value and ( not Allow ) ) or ( New_v and Allow );
          Port[$3D4] := Number; Port[$3D5] := Value;
          Sti;
          End;

{ Процедура устанавливает один из регистров контроллера графики.
Входные параметры такие же, как и у процедуры SetSinchronReg }
Procedure SetGraphReg ( New_v, Allow, Number : byte );
     var
          Value : byte;
     Begin
          Cli;
          Port[$3CE] := Number; Value := Port[$3CF];
          Value := ( Value and ( not Allow ) ) or ( New_v and Allow );
          Port[$3CE] := Number; Port[$3CF] := Value;
          Sti;
     End;

{ Процедура ожидает вертикальный обратный ход луча }
Procedure WaitVerticalRetrace;
     Begin
          While ( Port[$3DA] and $08 ) = 0 do;
     End;

{ Процедура копирует 0-й бит из "Allow" в бит XOR_17 }
Procedure XOR_17 ( Allow : byte );
     Begin
          Cli;
          Port[$3C2] := ( Port[$3CC] and $0DF ) or (( Allow and 1 ) shl 5 );
          Sti;
     End;

{ Процедура устанавливает регистр начального стартового адреса
контроллера ЭЛТ }
Procedure StartAddress ( Address : word );
     Begin
          SetEltControl ( Hi ( Address ), $0FF, $00C ); { установка старшего
                                                          байта адреса }
          SetEltControl ( Lo ( Address ), $0FF, $00D ); { установка младшего
                                                          байта адреса }
     End;

{ Процедура маскирует битовые плоскости видеопамяти VGA для
записи }
Procedure MapMask ( Mask : byte );
     Begin
          SetSinchronReg ( Mask, $0F, 2 );
     End;

{ Процедура выбирает битовую плоскость адаптера для чтения }
Procedure SelectPlane ( Number : byte );
     Begin
          SetGraphReg ( Number, 3, 4 );
     End;

{ Процедура устанавливает карту памяти.
 0 - A000h ( 128K адресного пространства )
 1 - A000h ( 64K адресного пространства )
 2 - B000h ( 32K адресного пространства )
 3 - B800h ( 32K адресного пространства ) }
Procedure MemoryMap ( Map : byte );
     Begin
          SetGraphReg ( Map shl 2, $0C, 6 );
     End;

{ Процедура устанавливает (Allow = 1) / сбрасывает (Allow = 0)
режим "цепочка 4" }
Procedure Chain_4 ( Allow : byte );
     Begin
          SetSinchronReg ( Allow shl 3, $08, 4 );
     End;

{ Процедура устанавливает (Allow = 1) / сбрасывает (Allow = 0)
режим "двойное слово" ЭЛТ }
Procedure DubleWordMode ( Allow : byte );
     Begin
          SetEltControl ( Allow shl 6, $40, $14 );
     End;

{ Процедура устанавливает
(Allow = 1) - режим байтов / (Allow = 0) - режим слов контроллера ЭЛТ }
Procedure ByteOrWordMode ( Allow : byte );
     Begin
          SetEltControl ( Allow shl 6, $40, $17 );
     End;

{ Процедура устанавливает (Allow = 1) / сбрасывает (Allow = 0)
свертку адреса контроллера ЭЛТ }
Procedure AddressConvolution ( Allow : byte );
     Begin
          SetEltControl ( Allow shl 5, $20, $17 );
     End;

{ Процедура устанавливает режим доступа ЦП к видеопамяти }
Procedure SetDMode ( Mode_Number : byte );
     Begin
          WaitVerticalRetrace; { ожидание обратного хода луча }
```

Листинг (продолжение)

```
        MapMask ( $0F );   { все плоскости разрешены }
        MemoryMap ( 0 );   { адрес видеобуфера A000h:0, 256Кб }
        Cli;
        Case Mode_Number of
            Mode_D0 : Chain_4 ( 0 ); { выбор режима доступа D0 }
            Mode_D4 : Chain_4 ( 1 ); { выбор режима доступа D4 }
        End;
        Sti;
    End;

{ Процедура установки циклического сдвига адреса, которая
используется ЭЛТ при выводе на экран }
Procedure SetSMode ( Mode_Number : byte );
    Begin
        WaitVerticalRetrace;
        Cli;
        Case Mode_Number of
            Mode_S0 : Begin
                        ByteOrWordMode ( 1 );
                        DubleWordMode ( 0 );
                      End;
            Mode_S2 : Begin
                        ByteOrWordMode ( 0 );
                        AddressConvolution ( 1 );
                        DubleWordMode ( 0 );
                      End;
            Mode_S4 : Begin
                        ByteOrWordMode ( 0 );
                        DubleWordMode ( 1 );
                      End;
        End;
        Sti;
    End;

{ Процедура установки комбинации режимов Sx, Dx }
Procedure Set_Mode_256 ( NewMode : byte );
    Begin
        Case NewMode of
            Mode_R0 : Begin
                        SetSMode ( Mode_S0 );
                        SetDMode ( Mode_D0 );
                      End;
            Mode_R4 : Begin
                        SetSMode ( Mode_S4 );
                        SetDMode ( Mode_D4 );
                      End;
        End;
        Mode_256 := NewMode;
    End;

var
    Active_Ptr : pointer; { указатель на активную страницу видеопамяти }
    Active_Ptr4 : Screen4PtrType absolute Active_Ptr;
    Active_Ptr0 : Screen0PtrType absolute Active_Ptr;

{ Процедура установки активной для вывода страницы }
Procedure SetActivePage ( Page : byte );
    Begin
        Case Mode_256 of
            Mode_R0 : Active_Ptr := Ptr ( $A000 + Page * $400, 0 );
            Mode_R4 : Begin
                        XOR_17 ( not ( ( Page shr 1 ) and 1 ) );
                        Active_Ptr := Ptr ($A000+(Page and 1) shl 12, 0);
                      End;
        End;
    End;

{ Процедура установки видимой страницы на экране }
Procedure SetVisualPage ( Page : byte );
    Begin
        WaitVerticalRetrace;
        StartAddress ( Page * $4000 );
    End;

const
    Last_Key : word = 0;

{ Процедура ожидания клавиши }
Procedure WaitKey; assembler;
    asm;
        mov ax, 0
        int 16h
        mov Last_Key, ax
    end;

{ Процедура "прокручивает" страницы на экране }
Procedure Visual_1;
    var
        I : word;
    Begin
        For I := 0 to 3 do Begin
            SetVisualPage ( I );
            WaitKey;
        End;
    End;

{ Процедура заполняет активную страницу цветом "Color" }
Procedure FillScreen ( Color : byte );
    Begin
        MapMask ( $0FF );
        Case Mode_256 of
            Mode_R0 : FillChar ( Active_Ptr0^, SizePage_R0, Color );
            Mode_R4 : FillChar ( Active_Ptr4^, SizePage_R4, Color );
        End;
    End;

{ Процедура вывода одного пиксела с координатами X, Y цветом "Color" }
Procedure PutPixel ( X, Y : integer; Color : byte );
    Begin
        Case Mode_256 of
            Mode_R0 : Begin
                        MapMask ( 1 shl ( X and 3 ) );
                        Active_Ptr0^[Y, X shr 2] := Color;
                      End;
            Mode_R4 : Active_Ptr4^[Y, X] := Color;
        End;
    End;

{ Функция чтения одного пиксела с координатами X, Y, возвращает его цвет }
Function GetPixel ( X, Y : integer ) : byte;
    Begin
        Case Mode_256 of
            Mode_R0 : Begin
                        SelectPlane ( X );
                        GetPixel := Active_Ptr0^[Y, X shr 2];
                      End;
            Mode_R4 : GetPixel := Active_Ptr4^[Y, X];
        End;
    End;

{ Процедура выполняет простейший тест режима "Number_Mode", (R0 или R4) }
Procedure Check_PROC ( Number_Mode : byte );
    var
        I : byte;
    Begin
        { установка режима "Number_Mode" }
        Set_Mode_256 ( Number_Mode );
        { заполнение всех 4 страниц разными цветами }
        For I := 0 to 3 do Begin
            SetActivePage ( I );
            FillScreen ( 2 * I + 1 );
        End;
        { визуальная прокрутка 4 страниц }
        Visual_1;
    End;

{ начало основной программы }
var
    I, J : integer;
BEGIN
    SetBIOSMode ( $13 );    { установка видеорежима BIOS 13h
                              ( разрешение 320 * 200, 256 цветов ) }
    Check_PROC ( Mode_R0 ); { проверка режима R0 }
    Check_PROC ( Mode_R4 ); { проверка режима R4 }
    { Демонстрация PutPixel на 1-й странице вывода }
    SetActivePage ( 1 );
    SetVisualPage ( 1 );
    For J := 50 to 100 do
        For I := 50 to J do PutPixel ( I, J, I );
    WaitKey;
END.
{ конец программы }
```

Только в МИКРОИНФОРМ

ЛЕКСИКОН Версия 1.2

Новая версия - Новые возможности:

- Ввод и редактирование многоязычных текстов - несколько языков в одном документе.
- Математические, химические и греческие символы на экране и принтере.
- Поддержка экрана VGA.
- Работа с «мышью».
- Работа в локальной сети.
- Обработка текстовых файлов неограниченного размера.
- Неограниченное число шрифтов.
- Перезагрузка драйверов принтера в процессе печати документа.
- Пропорциональная печать (в том числе многоколонная).
- Поддержка матричных и лазерных принтеров.
- Просмотр документа в уменьшенном виде (preview).
- Арифметический калькулятор.
- Конверторы файлов в форматы других текстовых редакторов и в формат Xerox Ventura Publisher.

Ждем Вас!

МИКРОИНФОРМ:

Россия
113184
Москва ул. Островского д. 44

Телефон: (095) 233 0006
Телефакс: (095) 235 1053
Электронная почта: lexicon@micro.msk.su

Практика построения экспертных систем средствами оболочки VPX

Ю. Носков

Для тех, кто хочет ознакомиться с основами построения и использования экспертных систем (ЭС), можно рекомендовать оболочки — специальные программные средства, включающие в себя ряд компонентов ЭС. Оболочка ЭС позволит Вам быстро и эффективно написать и отладить систему, сосредоточив внимание на главной компоненте ЭС — базе знаний.

Оболочка VPX вполне подходит для этих целей. По мнению американских специалистов по рынку программных изделий, VPX является бесспорным лидером по соотношению эффективность/затраты. Оболочка предназначена для IBM PC совместимых компьютеров. Предъявляет минимальные требования к составу аппаратуры машины: может работать как с жестким диском, так и без него, необходимый объем оперативной памяти — примерно 200 Кбайтов.

В состав VPX входят следующие программные модули и другие файлы (в скобках приведены объемы памяти в байтах):

- vpx.exe (29696) — головной модуль;
- vpxm.exe (70144) — подсистема трансляции;
- vpxi.exe (33792) — подсистема заполнения базы знаний;
- vpxt.exe (21504) — подсистема трассировки;
- vpxh.exe (20480) — справочная подсистема;
- vpxhelp.dbf (9390) — база данных справочной подсистемы;
- vpxhelp.hlp (13105) — база знаний справочной подсистемы;
- vpxhelp.txt (52080) — тексты рекомендаций справочной подсистемы.

Итак, все программное обеспечение оболочки и вспомогательные файлы легко умещаются на одной дискете.

В своем оригинальном виде система не допускала символы кириллицы в составе имен переменных и в меню их значений. Сейчас этот недостаток преодолен, для чего пришлось частично заменить встроенную таблицу кодов в файле VPXM.EXE.

Основы языка VPX

Если Бейсик можно выучить за месяц, то овладеть основами VPX вполне возможно за один день. В примере, приводимом ниже, использованы все предложения языка, необходимые для построения несложной, но работоспособной ЭС.

Любая экспертная система строится на основе знаний, определяющих область ее применения. Давайте построим нашу первую ЭС, основываясь на знаниях, которые есть у каждого в области предсказания погоды. Будем считать, что фирма Knowledge InterGalactic заказала нам систему для своих пилотов, собирающихся посетить Землю и еще не научившихся определять, нужно ли сегодня, покидая свой космический корабль, иметь с собой зонтик. В примере 1 приводится текст простой ЭС, предназначенной для этой цели.

В начале текста находятся комментарии. Признаком комментария в VPX служит восклицательный знак.

Из листинга видно, что ЭС на VPX состоит из трех разделов: установка параметров(1), сценарий(2) и база знаний(3), причем первый раздел может отсутствовать, а второй может состоять из нескольких предложений. Основу ЭС составляет база знаний, которые формулируются в виде правил вида ЕСЛИ...ТО...ИНАЧЕ... (IF... THEN...ELSE...).

Объекты, с которыми оперирует экспертная система, называются ЛИНГВИСТИЧЕСКИМИ ПЕРЕМЕННЫМИ, в нашей ЭС это Зонтик, Дождь, Гром, Будет_дождь и другие. На основе анализа конкретной ситуации система определяет наиболее разумное значение одной или нескольких таких переменных. В наших правилах имеются следующие значения переменных: Ну-

Пример 1. Текст простой экспертной системы на VPX

```
! Данная система предназначена для выдачи рекомендаций
! о необходимости взять зонтик.
! Исключительно для фирмы Knowledge InterGalactic !!!
!*********************************************
!      ** РАЗДЕЛ ПЕРВЫЙ: УСТАНОВКА ПАРАМЕТРОВ **
!*********************************************

!*********************************************
!      ** РАЗДЕЛ ВТОРОЙ: СЦЕНАРИЙ **
!*********************************************
actions
find Зонтик
display      "На основе Ваших ответов, проанализировав ситуацию,
       думаю, что {Зонтик}";
!*********************************************
!      ** РАЗДЕЛ ТРЕТИЙ: БАЗА ЗНАНИЙ **
!*********************************************
rule 1
if Дождь = Идет
then Зонтик = нужен_зонтик;
rule 2
if Будет_дождь = Да
then Зонтик = нужен_зонтик
else Зонтик = не_нужен_зонтик;
rule 3
if Гром = Гремит
then Будет_дождь = Да;
rule 3
if Небо = Пасмурно
then Будет_дождь = Да;
ask Дождь: "Идет ли дождь?";
choices Дождь: Идет,Не_идет;
ask Небо: "Небо сейчас пасмурно?";
choices Небо: Пасмурно,Ясно;
rule 4
if Небо = Ясно
then Будет_дождь = Нет;
rule 5
if лилии = Поникли
then Будет_дождь = Да
else Будет_дождь = Нет;
ask лилии: "Поникли лилии у Вас в саду?";
choices лилии: Поникли,Нет;
```

жен_зонтик, Не_нужен_зонтик, Гремит и так далее. Среди всего множества переменных одна или несколько должны являться переменными цели. Вспомним, что наша система предназначена для определения необходимости взять зонтик, поэтому целевая переменная у нас — Зонтик. Имя целевой переменной задается в разделе 2 с помощью предложения FIND.

Важно отметить связь целевой переменной с другими лингвистическими переменными. Если целью системы является определение значения переменной Зонтик, то система вначале предполагает, что искомым значением является Взять, поскольку правило 1 находится в начале текста базы знаний. Для подтверждения данной гипотезы система будет искать способ определения значения переменной Дождь, так как правило 1 устанавливает связь между этими переменными. Вначале система попытается найти правило, с помощью которого можно определить такое значение. Если это не удается, ищется текст вопроса (предложение ASK) для искомой переменной. В нашем случае такой вопрос имеется и система выдаст на экран тот текст, который записан в кавычках, после чего предъявит пользователю меню с вариантами ответов, содержащимися в предложении CHOICES. В дальнейшем поведение системы, наверное, очевидно. Если пользователь через меню выберет ответ Идет, то на основании правила 1 система будет считать, что переменная цели определена и логический вывод завершен. В противном случае вывод будет продолжен: будут анализироваться новые правила и задаваться новые вопросы.

В предыдущем абзаце мы использовали термин "система". Теперь настало время уточнить, что имелось в виду. Здесь описана логика работы важной части ЭС — системы вывода (во многих книгах по искусственному интеллекту она называется также машиной вывода). Одно из достоинств оболочек ЭС в том, что в них входит система вывода и для разработки системы достаточно написать только базу знаний и несколько дополнительных предложений и операторов языка. Алгоритм логического вывода, рассмотренный выше, известен под названием поиска в глубину.

Отметим следующие синтаксические особенности языка базы знаний:

- каждое правило начинается с ключевого слова RULE, после которого записывается номер или идентификатор правила;
- идентификаторы правил могут быть любыми, могут повторяться и на работу системы влияния не оказывают;
- можно произвольно испольэовать заглавные и строчные буквы в ключевых словах, например, можно написать rule, IF, tHEN;
- система различает заглавные и строчные буквы в именах и в значениях переменных, то есть Зонтик и ЗОНТИК — это разные переменные; идет и ИДЕТ — разные значения;
- в идентификаторах правил, именах и значениях переменных должно быть не более 20 произвольных символов; пробелы не допускаются.

Читатель может самостоятельно продолжить развитие базы знаний нашей первой ЭС, сформулировав любые правила, которые отображают его знания о том, как предсказать, будет ли сегодня дождь. Для переменных, значения которых можно установить на основании диалога с пользователем, нужно написать тексты вопросов и варианты ответов для показа в меню. Отметим, что человек, знания которого формулируются в базе знаний ЭС, играет роль эксперта.

Кроме базы знаний, простая экспертная система на VPX может иметь всего три предложения: ACTIONS, FIND и DISPLAY. Первое из них устанавливает начало второго раздела, именуемого "сценарий". В предложении FIND указывается имя переменной цели. В текст, написанный в предложении DISPLAY, нужно поместить текст сообщения, выводимого на экран. В текст обычно включается имя переменной цели в фигурных скобках, и система выводит в соответствующем месте текста полученное значение этой переменной. Текст ЭС можно ввести в компьютер как обычный текстовый файл с помощью любого редактора. Для VPX такие файлы должны иметь расширение KBS.

Установка оболочки VPX и работа с ней

Если Вы желаете установить оболочку на винчестер, нужно скопировать все указанные файлы в одну директорию. Впредь будем предполагать, что данная система установлена в директории VPX. Кроме перечисленных файлов, понадобится еще текстовый ре-

дактор. Редактор, входящий в состав стандартного пакета, неудобен, и его легко заменить на другой. Для связи с текстовым редактором в директории VPX нужно создать файл с именем EDNAME, в тексте которого указать имя файла, содержащего редактор. Например, если Вы захотите использовать текстовый редактор Norton editor, находящийся на диске D в директории NC, то файл EDNAME должен содержать одну строку: D:\NC\NE.COM

Возможно, Вы привыкли работать с текстовым редактором Лексикон. Можно подключить и его, но для нормальной работы потребуется скопировать некоторые его файлы в директорию VPX: это связано с особенностями данного редактора. Строка, содержащаяся в файле EDNAME, должна в этом случае иметь вид LEX.EXE.

Теперь можно приступать к работе с оболочкой, введя команду VPX.

После вызова системы на экране появляется заставка, в нижней части которой находится меню с такими позициями:

1.Help 2.Induce 3.Edit 4.Consult 5.Tree 6.Path 7.Filename 8.Quit

Для выбора позиции можно пользоваться клавишами перемещения курсора и клавишей Enter; можно также нажать на клавишу с соответствующей цифрой или на функциональную клавишу. Так, для получения помощи (Help) можно нажать Enter или 1, или клавишу F1.

Логика работы оболочки следующая. Мы можем выбрать любую из функций меню: 2, 3 или 4, и система попросит ввести имя файла ЭС. Если в текущей директории есть файлы с расширением KBS, то будет предъявлено меню для выбора любого из этих файлов. При желании можно перейти к обработке другого файла (функция 7) или указать путь (path) к другому диску (или директории).

Предположим, что наша ЭС введена в директорию VPX через текстовый редактор и находится в файле ZONT.KBS. После вызова оболочки выберем Consult и в появившемся меню — позицию ZONT. Если в тексте допущена синтаксическая ошибка, указывается номер неправильно введенной строки и автоматически вызывается текстовый редактор. Исправив ошибку, выйдем из редактора и сразу же снова окажемся в среде оболочки. Когда после нескольких таких итераций текст будет синтаксически правильным, на экране появится сообщение: FILE LOADED и мы получим возможность консультации со своей экспертной системой. Для этого в новом меню, появившемся в нижней части экрана, выберем позицию Go, будем отвечать на вопросы и получим совет. После выбора в меню варианта ответа нужно каждый раз нажимать на клавишу End. Обратите внимание на два нижних окна экрана. В них VPX показывает ход логического вывода, что помогает при отладке знаний.

Более сложная система

Очень скоро Вы обнаружите, что наша первая система работает не очень хорошо. Ее дальнейшее развитие возможно после знакомства с другими предложениями и операторами языка VPX. В примере 2 приводится текст усовершенствованной системы.

Прежде всего обратите внимание на базу знаний. В тексты правил введены дополнительные части с ключевым словом CNF и цифрой. Это — ФАКТОРЫ ДОВЕРИЯ, то есть некоторые оценки достоверности заключений, которые система вывода должна сделать на основе анализа ситуации при применении данного правила. Фактор доверия — целое число от 0 до 100. При выполнении логического вывода система ищет подтверждения гипотезам, связанным со значениями переменной цели, и присваивает найденным значениям оценки следующим образом. Сначала на основе правила 1 она выдвинет гипотезу, что значение переменной "зонт" равно "взять". Для проверки этой гипотезы нужно установить, каково значение переменной "дождь". Для этой переменной в базе знаний нет правил, но есть текст вопроса в операторе ASK. Значит, будет задан вопрос "Идет ли дождь?". Предположим, Вы ответили "Нет", тогда система будет пытаться подтвердить ту же гипотезу на основе других правил. Вы видите, что теперь она будет пытаться

Пример 2. Усовершенствованная экспертная система

```
! Экспертная система ЗОНТИК++
! Данная система предназначена для выдачи рекомендаций
! о необходимости взять зонтик.
! Исключительно для фирмы Knowledge InterGalactic !!!
!*********************************************
!      ** РАЗДЕЛ ПЕРВЫЙ: УСТАНОВКА ПАРАМЕТРОВ **
!*********************************************
endoff;
runtime;
execute;
plural: будет_дождь,зонт;
!*********************************************
!      ** РАЗДЕЛ ВТОРОЙ: СЦЕНАРИЙ **
!*********************************************
actions
truththresh = 60
find зонт
sort зонт
pop зонт,z
display "На основе Ваших ответов думаю, что {z} зонтик~";
!*********************************************
!      ** РАЗДЕЛ ТРЕТИЙ: БАЗА ЗНАНИЙ **
!*********************************************
ask лилии: "Поникли лилии у Вас в саду ?";
ask коровы: "Коровы стоят на лугу?";
choices дождь,гром,лилии,коровы: Да,Нет;
ask гром: "Гремит ли гром ?";
ask дождь: "Идет ли дождь ?";

rule 1
if дождь = Да
then зонт = нужно_взять;
rule 2
if дождь = Нет
and будет_дождь = Да
then зонт = нужно_взять;
rule 3
if дождь = Нет
and будет_дождь = Нет
then зонт = можно_не_брать
else зонт = может_пригодиться CNF 60;

rule 4
if гром = Да
then будет_дождь = Да CNF 70
else будет_дождь = Нет CNF 20;
rule 5
IF лилии = Да
then будет_дождь = Да CNF 55
else будет_дождь = Нет CNF 45;
rule 6
IF коровы  = Да
then будет_дождь = Да CNF 50
else будет_дождь = Нет CNF 25;
```

определить значение переменной "будет_дождь" и с этой целью на основе правила 4 — значение переменной "гром". Предположим, Вы ответили "нет" на вопрос о громе. Можно ли считать, что это означает (как это было в нашей предыдущей системе), что цель достигнута? Нет, поскольку правило 4 утверждает, что если гром гремит, то будет дождь (фактор доверия 70), а если не гремит, то дождя не будет (фактор доверия 20). Система запоминает значение будет_дождь=Нет CNF 20 и переходит к следующим правилам. Если она убедится, что лилии не поникли, то это будет дополнительным подтверждением той же гипотезы с фактором доверия 45. Для комбинирования факторов доверия применяется формула: CNF2=CNF0+(1-CNF0)*CNF1, где CNF0 — мера доверия к справедливости гипотезы, существовавшая до подтверждения ее с помощью данного правила; CNF1 — мера доверия, содержащаяся в данном правиле; CNF2 — результирующая мера доверия. Все CNF в указанной формуле имеют значения от 0 до 1, то есть уменьшены в 100 раз по отношению к принятым в тексте базы знаний. Таким образом, новое значение CNF в нашем примере будет 0,56. Формула построена так, что результат всегда получается в пределах от 0 до 1.

Для того чтобы система вывода хранила в процессе работы все возможные варианты гипотез о значениях переменных вместе с их CNF, нужно перечислить имена соответствующих переменных в предложении PLURAL, как это сделано в нашей системе для переменных "будет_дождь" и "зонт". Для каждой такой переменной выделяется стек. Чтобы после нахождения результата получить правильный ответ, нужно выполнить сортировку (оператор SORT) в стеке переменной цели. В результате сортировки все полученные значения будут располагаться в порядке убывания их CNF. Оператор POP извлекает из стека верхнее значение и передает его другой переменной (у нас — переменной z). Оператор с трудным для написания ключевым словом TRUTHTHRESH устанавливает порог доверия — величину, начиная с которой гипотеза считается достоверной.

Остальные операторы носят служебный характер: ENDOFF указывает, что ответ на вопрос считается введенным сразу после нажатия клавиши Enter; при его отсутствии нужно еще нажимать End; RUNTIME переводит экспертную систему в полноэкранный режим, без него на экране при консультации появляются, кроме основного окна диалога, еще два, показывающие процесс вывода; EXECUTE сообщает оболочке, что нужно приступать к консультации сразу после загрузки системы в память, не показывая дополнительного меню, в котором для консультации нужно выбрать Go.

В рамках данной статьи рассмотрены лишь некоторые наиболее важные для начала работы предложения языка и возможности оболочки VPX. На основе их можно построить и отладить свою собственную экспертную систему. Читатели, желающие более глубоко изучить VPX и другие средства построения ЭС, могут обращаться к автору через редакцию журнала.

В заключение хотелось бы выразить признательность студенту МИЭМ А.В.ПОМЕРАНЦЕВУ, выполнившему русификацию оболочки VPX. ■

Спиок рекомендуемой литературы

1. Левин Р., Дранг Д., Эделсон Б. Практическое введение в технологию искусственного интеллекта и экспертных систем с иллюстрациями на Бейсике.- М.: Финансы и статистика, 1990.
2. Нейлор К. Как построить свою экспертную систему.- М.: Энергоатомиздат, 1991.

Сегодня S P I R I T представляет

ArcView 6.3

Просмотр и Редактирование Архивов

Экономия с удовольствием

— Распознавание архивов ARJ, LZH, ICE, ARC, PAK, ZIP, CHZ, и SFX
— Быстрее и удобнее, чем оригинальный архиватор
— Распаковка, просмотр, тестирование и удаление файлов в архиве
— Работа с поддиректориями в архивном файле
— Множество других функций всего за 475 рублей

Тысячи копий уже приносят пользу

Сегодня S P I R I T представляет

FileCrypter 2.0

Простая система для профессиональной криптографической защиты коммерческих данных для PC

Надежнее, чем сейф

Разнообразные алгоритмы шифрования в одной среде с удобным интерфейсом:

— Надежный DES (Федеральный криптостандарт США)
— Быстрый RSC (оригинальный отечественный алгоритм)
— Надежный и быстрый ГОСТ (Федеральный криптостандарт России)
— Цифровая подпись

Цена всего 4.875 рублей за копию

Информация о новых вирусах в формате антивирусной программы AVSP 2.0.

AVSP предоставляет пользователю возможность непосредственного ввода информации о новых вирусах в программу. Благодаря регулярной публикации такой информации AVSP на Вашем компьютере не отстанет от времени и создателей новых вирусов.

AVSP содержит ряд средств борьбы с неизвестными вирусами. ДИАЛОГ-МГУ обращается ко всем читателям с просьбой присылать информацию об обнаруженых новых вирусах.

Для ввода приведенной ниже информации в Вашу программу войдите в пункт меню "Данные о файлах и вирусах", подменю "Изменение информации о вирусах". Для более подробного описания можно обратиться к документации (пункт "Ввод информации о вирусах из журналов").

Имя вируса: Tumen
Номер: 223.1-49124
Тип вируса: PROGRAM
Размер: 1663
Шаблон: 812E12000001030603000580008BD0
Процедуры: *.COM { Opt(End:-1214) Mov(MP:-449,Beg:0,3) Tr(MP:-449) }
Справка: Создан, по словам авторов, в Тюмени в 1990 году. Содержит много текста типа "Full Moon tonight... The Time of Wolfbacks" и т.п. Предупреждает, что это только ДЕМО-версия (v0.5) вируса...

Имя вируса: MSU-3125
Номер: 224.1-27058
Тип вируса: PROGRAM
Шаблон: 51509CBB7502B9250C2BCB2EA0DE01
Процедуры: *.EXE { Opt(End:-2629) Mov(MP:54,Head:20,4) SubW(Head:10,$103) TrEXE(End:-3110) }
*.COM { Opt(End:-2629) Mov(End:-2843,Beg:0,3) Tr(End:-2850) }
Справка: У зараженных EXE файлов длина увеличивается от 3110 до 3125 байт. У COM файлов на 2850 байт. Случайным образом заражаются файлы при входе в директорию, а также при запуске и обращении к программным файлам. Обнаружен в МГУ.

Внимание !!! Новая информация по вирусу IBM, который уже есть в базе данных AVSP:

Имя вируса: IBM
Номер: 130.2-1814
Тип вируса: PROGRAM
Размер: 547
Шаблон: 8ED88B1620022E891600018A362202
Процедуры: *.COM { Opt(End:-537) Mov(MP:526,Beg:0,3) Tr(MP:-10) }
Справка: Содержит строку "IBM".

По вопросам приобретения AVSP 2.0 обращайтесь в ДИАЛОГ-МГУ по телефонам (095) 939-38-90, 939-38-91. Подробная информация высылается бесплатно.

Цена одного комплекта AVSP 2.0 на 28 октября 1992 г. составляет 1924 рублей (с учетом 28% налога). При покупке нескольких комплектов предоставляется скидка. Скидка для дилеров - до 60%.

Уважаемые читатели!

Мы предлагаем Вашему вниманию

компьютерный "ларек".

Мы преследуем две основные цели:

— дать возможность отечественным разработчикам инструментальных средств выйти на российский рынок программного обеспечения;

— обеспечить как можно более широкий круг читателей недорогими и оригинальными средствами разработки программного обеспечения.

Код	Продукт	Краткая характеристика	Объем поставки	Цена, $	Демо	Установщик	Требования к оборудованию	Сопровождение пользователей	Число продаж	Скидка по upgrade	Скидки в оплате	Примечания
SL-01	Turbo Games	Объектно-ориентированная библиотека для Turbo Pascal 6.0. Спрайты, звуковые эффекты, трехмерная графика, стереоэффекты. Доступ к ZIP файлам.	2 дискеты + документация	12 для индивидуальных покупателей, 25 для организаций	+	+	EGA адаптер, IBM PC/AT	+		+	+	Готовится версия для VGA
SL-02	Data Vision 1.5	Объектно-ориентированная библиотека для Turbo Pascal 6.0. Создание и управление реляционными базами данных. Поддерживаются DBF и DB форматы. Поддержка электронных таблиц.	2 дискеты + документация	12	+	—	IBM PC/XT	+	37	+	—	Версия 2.0 (осень 93 года) будет поддерживать файлы СУБД Clarion и содержать генератор отчетов
IS-01	HyperRef	Инструментальнсе средство для создания гипертекстовых систем. Поддержка текстовой информации, изображений и исполняемых программ.	2 дискеты по 1,2 Мб	35	+	—	IBM PC/AT, EGA/VGA, 3 Мб свободного пространства на HDD, 350 Кб ОЗУ--просмотр документа,450 Кб – редактирование документа.					
CS-01	QuickTools V1.0	Средство разработки диалога в Paradox.	1 дискета 1,2 Мб или 1,44 Мб	10	+	+	IBM PC/AT. EGA/VGA, 3 Мб на HDD, 1-2 Мб ОЗУ.		+			
UT-01	ArcView 6.1	Просмотр и редактирование заголовков архивных файлов с возможностью подключения собственных типов архивов.	1 дискета 360 Кб	1	—	+	IBM PC/AT, Hercules, EGA/VGA	+				
SE-01	Crypter 2.0	Система шифрования файлов. Поддерживаются алгоритмы DES, ГОСТ	1 дискета 360 Кб	12	—	+	IBM PC/AT, Hercules, EGA/VGA	+				
IS-02	OKO 1.0	Инструментальное средство для разработки приложений, интерфейс сходен с интерфейсом электронной таблицы. Позволяет быстро переносить документы с PC и объединять их в единую систему обработки данных. Реализовано как верхняя оболочка сетевой СУБД dbVISTA.	4 дискеты по 360 Кб или 1 дискета 1,2 Мб Документация: 2 файла на установочной дискете.	18	+	+	IBM PC/XT, EGA/CGA/VGA, 1 Мб HDD, 640 Кб ОЗУ	+		+ (18 $ за 1-ю установку и 3 за каждую последующую)	+ (20% для новых версий для пользователей OKO)	
SC-01	АвтоГРАФ	Пакет для создания на экране графиков с результатами обработки числовых данных и печати их на 9- и 24-игольчатых принтерах. Поддержка форматов ЕСКД.	1 дискета 360 Кб	10	+	+	IBM PC/XT, EGA/VGA, 500 Кб на HDD, ОЗУ – 200 Кб	+	70	+	+ для учебных заведений	Готовится версия 3.0

Если Вас заинтересовали приведенные здесь продукты или Вы хотите получить дополнительную информацию, звоните, пишите, приезжайте.

Оплата принимается только в рублях по текущему курсу.

Адрес для писем: 117419 Москва, а/я 765

Тел.: (095) 238-99-33 Факс: (095) 238-32-78

CA–Clipper/Compiler Kit

For dBASE IV

Компилируйте и распространяйте dBASE IV-программы с помощью CA-Clipper!

CA–Clipper Compiler Kit for dBASE IV создает из Ваших dBASE IV программ автономные выполняемые .EXE файлы, которые можно свободно распространять как на PC, так и в локальных сетях без дополнительных платежей или Runtime обеспечения.

Компиляция происходит автоматически. Скомпилированные программы полностью поддерживают функциональные возможности dBASE IV и приобретают преимущества, заложенные в CA–Clipper:

— оптимальное управление памятью;
— быстрое выполнение программ;
— доступ ко всем мощным языковым средствам CA–Clipper, включая классы объектов.

Системные требования:

* CA–Clipper (Версия 5.01)
* IBM PC XT, AT, PS/2 или 100% совместимый персональный компьютер
* жесткий диск
* минимум 640K RAM
* драйвер расширенной памяти LIM 3.2 или выше
* DOS 3.1 или выше

CA–Clipper/Compiler Kit For dBASE IV работает в сетях, совместимых с DOS 3.1 или выше

Только три простых шага необходимы для создания автономных программ.

1. Создаете прикладную программу, используя обычный инструментарий dBASE IV, включая генератор прикладных программ.
2. Оптимизируете и тестируете Вашу программу.
3. Набираете команду COMPILE имя файла

То, что Ashton Tate и Borland обещали шесть лет, Computer Associates предлагает Вам сегодня!

127108, Москва, ул. 2–я Ямская, д. 15
Телефоны: 289–44–77, 289–44–83

COMPUTER® ASSOCIATES
Software superior by design